

Nucleus μ iPLUS

μ ITRON4.0 Specification Standard Profile

Users Manual

English Version 1.05
[Temporary Draft]



μITRON4.0 Specification is an open architectural Real Time Kernel Specification, laid down by the **ITRON** Technical Committee of the **TRON** Association taking the lead. The **μITRON4.0** Specification could be downloaded from the ITRON Project web site (<http://www.itron.gr.jp/>).

- **TRON** is an abbreviation of “**The Real-time Operating system Nucleus**”.
- **ITRON** is an abbreviation of “**Industrial TRON**”.
- **μITRON** is an abbreviation of “**Micro Industrial TRON**”.
- **TRON**, **ITRON**, and **μITRON** are not the name that points out for specific product or specific product group.

Nucleus μiPLUS (μITRON4.0 Specification Standard Profile) Users Manual

Reproducing, copying and reprinting the part or all of the content of this manual without notice are forbidden.

The contents of this manual may be changed without notice.



Table of Contents

1.	Nucleus μ iPLUS.....	1
	Users Manual.....	1
1	Introduction.....	1
1.1	Introduction and Features.....	1
1.2	Relationship with Nucleus PLUS.....	1
1.3	Service Call from Non-Task Contexts.....	2
2	Start-up.....	3
2.1	Packaging Content.....	3
2.2	Compile and Link Procedures.....	4
2.3	Configuration File.....	5
2.4	System Initialization Procedure.....	5
2.5	Service call that could be used with Application_Initialize.....	6
2.6	Porting Item.....	6
2.6.1	Standard Profile.....	7
2.6.2	Defining Timer Tick.....	7
3	Functionality of the Kernel.....	8
3.1	Multi-task Function.....	8
3.2	Real Time Function.....	8
3.3	Task Management.....	8
3.3.1	Task Status.....	9
3.3.2	Scheduling Rule.....	9
3.3.3	Priority of the Task.....	10
3.4	Synchronous Function with Tasks.....	10
3.5	Task Exception Handling Function.....	10
3.6	Synchronization and Communication Function.....	11
3.6.1	Semaphore.....	11
3.6.2	Event Flag.....	12
3.6.3	Data Queue.....	12
3.6.4	Mailbox.....	13
3.7	Memory Pool Management Function.....	14
3.7.1	Fixed-sized Memory Pool.....	14
3.7.2	Variable-sized Memory Pool.....	14
3.8	Time Management Function.....	15
3.8.1	System Time Management.....	15
3.8.2	Cyclic Handler.....	15
3.9	System Status Management Function.....	16
3.9.1	Rotate Task Priority Order.....	16
3.9.2	CPU Lock.....	16
3.9.3	Dispatch Disable.....	16
3.10	Interrupt Management Function.....	16
3.11	System Configuration Management Function.....	17
4	Service Call Reference.....	17
4.1	Task Management Functions.....	18
4.1.1	CRE_TSK Create Task _Static API_S.....	18
4.1.2	cre_tsk Create Task.....	19
4.1.3	del_tsk Delete Task.....	19

4.1.4	acre_tsk	Create Task (ID number automatically assigned).....	21
4.1.5	act_tsk	Activate Task_S_.....	22
4.1.6	iact_tsk	Activate Task_Exclusive for Non-task Context_S_.....	23
4.1.7	can_act	Cancel Activation Request_S_.....	24
4.1.8	sta_tsk	Start a task with a start code.....	25
4.1.9	ext_tsk	Terminate the invoking task_S_.....	26
4.1.10	ter_tsk	Terminate a task_S_.....	27
4.1.11	chg_pri	Change a task's priority_S_.....	27
4.1.12	get_pri	Get task priority_S_.....	29
4.1.13	ref_tsk	Reference a task's state.....	30
4.1.14	ref_tst	Reference a task's state (minimal function).....	31
4.2	Task dependent synchronization function.....		32
4.2.1	slp_tsk	Put task to sleep_S_.....	32
4.2.2	tslp_tsk	Put task to sleep with a timeout_S_.....	33
4.2.3	wup_tsk	Wakeup a task_S_.....	34
4.2.4	iwup_tsk	Wakeup a Task (exclusive to Non-task context)_S_.....	35
4.2.5	can_wup	Cancel wakeup request_S_.....	36
4.2.6	rel_wai	Release task from WAITING state_S_.....	37
4.2.7	irel_wai	Release task from WAITING state_exclusive to Non-task context_S_.....	38
4.2.8	sus_tsk	Suspend a task_S_.....	39
4.2.9	rsm_tsk	Resume a suspended task_S_.....	40
4.2.10	frsm_tsk	Forcefully resume a suspended task_S_.....	41
4.2.11	dly_tsk	Delay the invoking task_S_.....	42
4.3	Task exception handling functions.....		43
4.3.1	DEF_TEX	Define a task exception handling routine (static API)_S_.....	43
4.3.2	def_tex	Define a task exception handling routine.....	44
4.3.3	ras_tex	Raise task exception handling_S_.....	45
4.3.4	iras_tex	Raise task exception handling (exclusive to Non-task context)_S_.....	46
4.3.5	dis_tex	Disable task exception handling_S_.....	47
4.3.6	ena_tex	Enable task exception handling_S_.....	48
4.3.7	sns_tex	Sense task exception handling status_S_.....	49
4.3.8	ref_tex	Reference state of task exception handling_S_.....	50
4.4	Synchronization and Communication Functions - Semaphore.....		51
4.4.1.1	CRE_SEM	Create Semaphore (Static API)_S_.....	51
4.4.2	cre_sem	Create Semaphore.....	52
4.4.3	acre_sem	Create Semaphore (ID number automatically assignment).....	53
4.4.4	del_sem	Delete Semaphore.....	54
4.4.5	sig_sem	Release Semaphore_S_.....	55
4.4.6	isig_sem	Release Semaphore_exclusive to Non-task context_S_.....	56
4.4.7	wai_sem	Get Semaphore_S_.....	57
4.4.8	pol_sem	Get Semaphore (polling)_S_.....	58
4.4.9	twai_sem	Get Semaphore (timeout)_S_.....	59
4.4.10	ref_sem	Refer Semaphore State.....	60
4.5	Synchronization and Communication Functions - Eventflag.....		61
4.5.1	CRE_FLG	Create an eventflag (Static API)_S_.....	61
4.5.2	cre_flg	Create an eventflag.....	62
4.5.3	acre_flg	Create an eventflag (ID number automatically assignment).....	63
4.5.4	del_flg	Delete eventflag.....	64
4.5.5	set_flg	Set Eventflag_S_.....	65
4.5.6	iset_flg	Set eventflag_exclusive to Non-task context_S_.....	66
4.5.7	clr_flg	Clear Eventflag_S_.....	67
4.5.8	wai_flg	Wait for Eventflag_S_.....	68
4.5.9	pol_flg	Wait for Eventflag (Polling)_S_.....	69
4.5.10	twai_flg	Wait for Eventflag with Timeout_S_.....	70
4.5.11	ref_flg	Reference Eventflag Status.....	71
4.6	Synchronization and Communication Functions – Data queue.....		72
4.6.1	CRE_DTQ	Create data queue (Static API)_S_.....	72
4.6.2	cre_dtq	Create data queue.....	73
4.6.3	acre_dtq	Create data queue (ID number automatically assignment).....	74

4.6.4	del_dtq	Delete data queue	75
4.6.5	snd_dtq	send to data queue_S_	76
4.6.6	psnd_dtq	send to data queue (polling)_S_	77
4.6.7	ipsnd_dtq	send to data queue (exclusive to Non-task context)_S_	78
4.6.8	tsnd_dtq	send to data queue (timeout)_S_	79
4.6.9	fsnd_dtq	forced send to data queue_S_	80
4.6.10	ifsnd_dtq	forced send to data queue_S_	81
4.6.11	rcv_dtq	receive data element from data queue_S_	82
4.6.12	prcv_dtq	receive data element from data queue (polling)_S_	83
4.6.13	trcv_dtq	receive data element from data queue (timeout)_S_	84
4.6.14	ref_dtq	reference the state of the data queue	85
4.7	Synchronization and Communication Functions – Mailbox		86
4.7.1	CRE_MBX	create mailbox (static API)_S_	86
4.7.2	cre_mbx	create mailbox	87
4.7.3	acre_mbx	create mailbox (ID number automatically assigned)_S_	88
4.7.4	del_mbx	delete mailbox	89
4.7.5	snd_mbx	send to mailbox_S_	90
4.7.6	isnd_mbx	send to mailbox (exclusive to Non-task context)	91
4.7.7	rcv_mbx	receive from mailbox_S_	92
4.7.8	prcv_mbx	receive from mailbox (polling)_S_	93
4.7.9	trcv_mbx	receive from mailbox (timeout)_S_	94
4.7.10	ref_mbx	reference state of mailbox	95
4.8	Memory Pool Management Functions Fixed-sized memory pools		96
4.8.1	CRE_MPF	Create fixed-sized memory pool (static API)_S_	96
4.8.2	cre_mpf	Create fixed-sized memory pool	97
4.8.3	acre_mpf	Create fixed-sized memory pool (ID number automatically assigned)	98
4.8.4	del_mpf	Delete fixed-sized memory pool	99
4.8.5	get_mpf	Get a fixed-sized memory block_S_	100
4.8.6	pget_mpf	Get a fixed-sized memory block. (polling)_S_	101
4.8.7	tget_mpf	Get a fixed-sized memory block. (timeout)_S_	102
4.8.8	rel_mpf	Release a fixed-sized memory block_S_	103
4.8.9	ref_mpf	Reference fixed-sized memory pool status	104
4.9	Memory Pool Management Functions Variable-sized Memory Pools		105
4.9.1	CRE_MPL	Create variable-sized memory pool (Static API)	105
4.9.2	cre_mpl	Create variable-sized memory pool	106
4.9.3	acre_mpl	Create variable-sized memory pool (automatic ID number assigned)	107
4.9.4	del_mpl	Delete variable-sized memory pool	108
4.9.5	get_mpl	Allocate a variable-sized memory block	109
4.9.6	pget_mpl	Allocate a variable-sized memory block (polling)	110
4.9.7	tget_mpl	Allocate a variable-sized memory block (timeout)	111
4.9.8	rel_mpl	Return variable-sized memory block	112
4.9.9	ref_mpl	Reference variable-sized memory pool status	113
4.10	System Time Management		114
4.10.1	set_tim	Set system time_S_	114
4.10.2	get_tim	Reference system time_S_	115
4.10.3	vget_tim	Reference system time	116
4.10.4	CRE_CYC	Create Cyclic handler (Static API)_S_	117
4.10.5	cre_cyc	Create Cyclic handler	118
4.10.6	acre_cyc	Create Cyclic handler (ID number automatically assigned)	119
4.10.7	del_cyc	Delete cyclic handler	120
4.10.8	sta_cyc	Start cyclic handler operation_S_	121
4.10.9	stp_cyc	Stop Cyclic handler operation_S_	122
4.10.10	ref_cyc	Reference the status of a cyclic handler	123
4.12	System state management function		133
4.12.1	rot_rdq	Rotation of priority of task_S_	133
4.12.2	irot_rdq	Rotation of priority of task (exclusive to Non-task context)_S_	134
4.12.3	get_tid	Reference of task ID under execution (exclusive to Non-task context)_S_	135
4.12.4	iget_tid	task ID Reference under execution_S_	136
4.12.5	vget_tid	Reference of the activating task ID (exclusive to Task Context)	137

4.12.6	loc_cpu	Shift to CPU lock state_S_	138
4.12.7	iloc_cpu	Shift to CPU lock state (exclusive to Non-task context)_S_	139
4.12.8	unl_cpu	Cancellation of CPU lock state_S_	140
4.12.9	iunl_cpu	Cancellation of CPU lock state (exclusive to Non-task context)_S_	141
4.12.10	dis_dsp	Disable dispatch_S_	142
4.12.11	ena_dsp	Enable dispatch_S_	143
4.12.12	sns_ctx	Sense context_S_	144
4.12.13	sns_loc	Reference CPU locked state_S_	145
4.12.14	sns_dsp	Reference dispatch disabled state_S_	146
4.12.15	sns_dpn	Reference dispatch pending state_S_	147
4.12.16	DEF_INH	Define interrupt handler (static API)_S_	148
4.12.17	def_inh	Define interrupt handler	149
4.12.18	DEF_EXC	Define CPU exception handler (static API)_S_	150
4.12.19	def_exc	Define CPU exception handler	151
4.12.20	ref_cfg	Reference configuration information	152
4.12.21	ATT_INI	Attach initialization routine (static API)_S_	153
5	Error during Runtime		154
5.1	System Error		154
6	Configurator		155
6.1	Environmental Variable		155
6.2	Pre-processor Directive		156
6.3	Static API		156
6.4	INCLUDE		156
6.5	Command Line Argument		156
6.6	Disable the Unnecessary Data Area for Management		157
6.7	User Setting of the Data Area for Management		158
6.8	Others		158
6.9	Configuration File Sample		159
7	Internal Structure		160
7.1	Object Management		160
7.2	Task Management		160
7.3	Delayed Execution		161
7.4	Ready Queue		162
7.5	Scheduling Function		162
7.6	Interrupt Behavior		163
7.7	Deterring Unnecessary Process		164
7.8	Internal Data Size		164
7.9	C++		164
7.9.1	Constructor of the Global Object		165
7.9.2	new/delete operator		165
Appendix			166
Nucleus µiPLUS			167

Diagram Contents

List of Tables

Table 3.3-1 Task Management Service Call.....	9
Table 3.4-1 Synchronous with Task Service Call.....	10
Table 3.5-1 Task Exception Handling Service Call	11
Table 3.6-1 Synchronization and Communication function, Semaphore Service Call.....	12
Table 3.6-2 Synchronization and Communication Event Flag, Service Call	12
Table 3.6-3 Synchronization and Communication function, Data Queue Service Call.....	14
Table 3.6-4 Synchronization and Communication function, Mailbox Service Call	14
Table 3.7-1 Fixed-sized Memory Pool Service Call.....	15
Table 3.7-2 Variable-sized Memory Pool Service Call.....	15
Table 3.8-1 System Time Management Service Call	15
Table 3.8-2 Cyclic Handler Service Call	20
Table 3.9-1 System Status Management Function, Service Call	216
Table 3.10-1 Interrupt Management Service Call.....	21
Table 3.11-1 System Configuration Management Service Call.....	21
Table 6.3-1 A list of Static API that could be used by the Configurator (Alphabetical Order)	156

List of Figures

Figure 0-1 The relationship between Nucleus μ iPLUS and PLUS.....	2
Figure 0-2 Compile and Link Procedure.....	Error! Bookmark not defined.
Figure 0-3 System Initialization Procedure	6
Figure 3.1-1 Benefits of Multi-tasking	8
Figure 3.3-1 Scheduling Rule.....	10
Figure 3.5-1 Task Exception Handling.....	11
Figure 3.7-1 Securing Fixed-sized Memory Block	14
Figure 3.7-2 Securing Variable-sized Memory Block	19
Figure 6.1 mip_CFGW.exe Start-up Window	155
Figure 6.2 Setting dialog of mip_CfgW.exe	155
Figure 7.1-1 How μ iPLUS function has been implemented.....	160
Figure 7.2-1 The Link between TCB and iTCB	161
Figure 7.4-1 Ready Queue.....	162

1 Introduction

1.1 Introduction and Features

Nucleus μ iPLUS has the following features:

- Compliant with **μ ITRON4.0** Standard Profile Specification
- Various service supported other than **μ ITRON4.0** Standard Profile Specification
- Highly Portable
- Small and fast
- Provides with wide range of functions
- Provides with Integrated Development Environment
- Configurator Tool available

95% of **Nucleus μ iPLUS** is written in **C** Language, and it is used as a library. In order to use **Nucleus μ iPLUS**, you have to create the application that uses the service calls regulated by the **μ ITRON4.0** standard specification, and link it to **Nucleus μ iPLUS** library. The execution image that has been created should be either downloaded or be implemented to the target, after loading into the **ROM**.

Nucleus μ iPLUS does not support switching function from user mode to **CPU**'s supervisor mode (or the Kernel mode), and also the **MMU** memory support function. Instead, we have adopted Flat Memory Model, which enables an equal access of all memory within the system. The first time users could easily chose the **OS (Operating System)** itself, because it is designed in such a way that the thread switching is fast and small. On the other hand, developers must be aware not to let memory leak occur when programming.

All other software components that are treated as an option, such as **TCP/IP** and Graphics, are provided in library format. Because of this reason, developers can easily add these functions to their existing systems using **Nucleus μ iPLUS**.

The users could easily port themselves, because **Nucleus μ iPLUS** is provided with the source code.

The defining of the resource is done by the data definition of the **C** Language's global variable. By using the Configuration Tool, the user could automatically create this definition.

1.2 Relationship with Nucleus PLUS

Nucleus μ iPLUS uses **Nucleus PLUS**, the leader of source code, no royalty **RTOS (Real Time Operating System)**, as the core. It has been designed in such a way that the **ITRON** specification interface library is been added in **Nucleus PLUS**'s own specification.

Since the specification design of **Nucleus PLUS** is similar to that of **ITRON**, the task condition is very similar. **ITRON** application could be developed when the developer uses this interface library. Also, the application could directly call **Nucleus PLUS** system calls, instead of using this **ITRON** library. The reason is because it enables the developers to use all of the software components such as **TCP/IP** protocol, File systems, and drivers that are designed to work with **Nucleus PLUS**. Certainly, coexisting of **Nucleus PLUS** task and descriptive task by **ITRON** is possible. This compatibility is maintained for some time, but when you use **PLUS** service calls for the user application, we recommend you to manage in order to pick up the applicable code later.

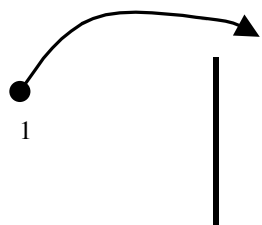
Furthermore, the Signal function of **Nucleus PLUS** is replaced by Task Exceptional function of **Nucleus μ iPLUS**. You have to be careful when you port the application that uses Signal function of **Nucleus PLUS**, because the behavior is slightly different.

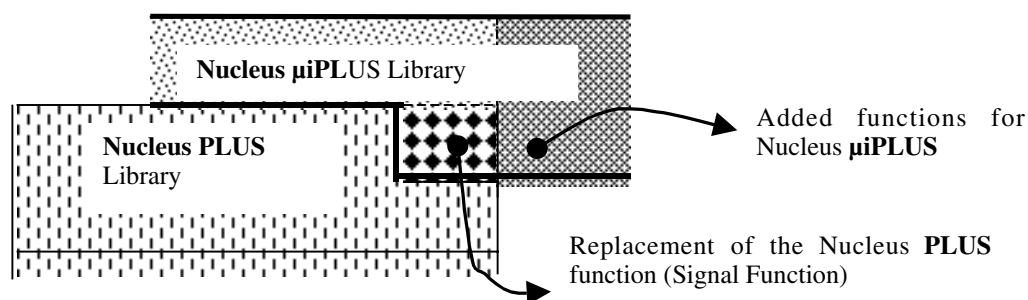
Direct access to **Nucleus PLUS** is possible,
for the compatibility to **Nucleus PLUS**
Library.
(**Nucleus NET**, **Nucleus FILE** etc.)



1

Interface Library to
Nucleus PLUS for
Nucleus μ iPLUS





The Start-up Code specified by the Static API in the configuration file, and to create **TASK_MAIN** are created.

Figure 1.2-1 The relationship between μPLUS and PLUS

1.3 Service Call from Non-Task Contexts

Contexts, or an operating environment that can be regarded as a part of a task process are generically called task contexts, while other contexts such as the Interrupt Handler and Exceptional Handler are generically called non-task contexts. Since **Nucleus μPLUS** does not have long interrupt disable area within its **OS**, the service call that could be called from the task context from the non-task context, and from both task and non-task context are differentiated.

Service calls that are exclusive to non-task context always start with a letter “**i**”. Moreover, this service call is executed with a delay, with some exceptions. In other words, the request of the service call that has been issued from non-task context will be withheld in the **OS**, and after the interrupt process is done, it is executed before the control gets back to the task. Because of this reason, the **OS** has a buffer to store the service calls that are exclusive to non-task context. When this buffer gets full (when accepting the request exceeds the processing of the request), the service call that starts with a letter “**i**” returns **E_NOMEM**. Also, the service calls that could be called by both task context and non-task context starts with “**sns**”.

2 Start-up

2.1 Packaging Content

Nucleus μ iPLUS products includes the following:

- (1) **Nucleus μ iPLUS** Source File
- (2) **Nucleus μ iPLUS** Batch File for creating Libraries
- (3) **Nucleus μ iPLUS** Sample Source File
- (4) **Nucleus μ iPLUS** Batch File for creating Samples
- (5) **Nucleus μ iPLUS** Configurator
- (6) **Nucleus μ iPLUS** Configurator **for Windows**

Manual

- (1) **Nucleus μ iPLUS** Users Manual

Though, **Nucleus μ iPLUS** supports various tools and **CPUs**, the **C** Language source file is used on every board in the same manner. However, please refer to the **Nucleus PLUS** Target Specific Note, because the assembler file and the batch file for build may vary by using different target **CPU** and tools.

For the standard users developing **ITRON** specific application using **Nucleus μ iPLUS**, the application development could be done by just referring to **μ iPLUS** related manuals. You do not necessarily have to refer to **Nucleus PLUS** manual.

2.2 Compile and Link Procedures

The procedure used to design the application is depicted below.

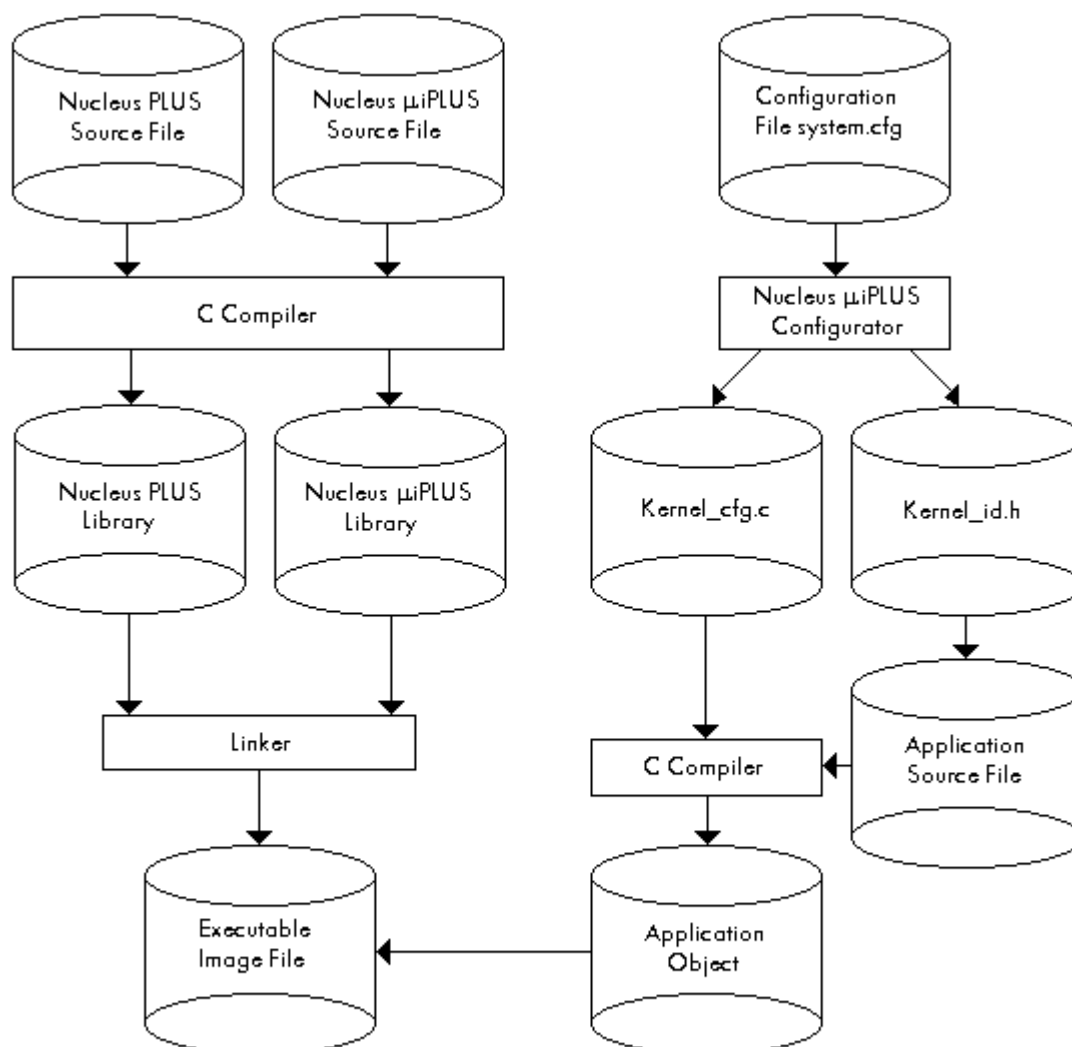


Figure 2.2-1 Compile and Link Procedures

First, the user has to create **Nucleus PLUS** library and **Nucleus μPLUS** library. This can be done by simply executing the attached batch file. The **Nucleus μPLUS** library needs to be rebuilt when the constant of the kernel indicated in section 2.6 Porting Item, is changed.

Next, the user has to determine which resources, such as task and semaphore, to be requested to the **OS**, and then has to create a configuration file. It may be best to first customize the sample. By loading this file to a Configurator, it will output the **C** source and Include file.

Application programmer creates the actual task process in a different file. By linking the kernel library, the configuration output module, and the user application, creates the execution image. Creating the library and using the configuration is needed only during the first stage, since usually the application file is the only thing that needs to be recompiled and modified.

2.3 Configuration File

It is possible to describe by using the static **API** in configuration file (**system.cfg**), in order to request the resources, such as the task to the kernel. The configurator outputs the **C** source (**kernel_cfg.c**) and the Include file (**kernel_id.h**) that constructs the kernel, depending upon the contents that are described in the configuration file. The output file includes the Start-up routine (for Nucleus **µiPLUS**, it is called the **Application_Initialize**). At least one task has to be created in this routine. Since it is a **C** source file, user can also customize by directly editing it.

It is helpful to include **kernel_id.h** that the Configurator outputs for specifying resources such as the tasks from the application module. Compiling and linking the output file, along with the file that includes the task descriptions creates the final target application. In the configuration file, added to the static **API**, preprocessor directive for C Language Management could also be described.

Configurator for **Windows** enables ease to specify configuration file and file output settings, and just by pushing the button, explanation of the configuration file, kernel construction, and output initialization file will become possible. Moreover, by the user adding the modification based on the output file, the Configurator could create kernel construction including the initialization management. Please refer to section 6 Configurator for the precise usage of Configurator and the description of configuration file.

Ex.) Define task by Configuration File (**system.cfg**)

```
#define STACK_SIZE 2048
#define PRIORITY_DEF 1
CRE_TSK(TASK_MAIN, { TA_HLNG|TA_ACT, 1, task_main, PRIORITY_DEF,
                    STACK_SIZE, NULL });
```

The Configurator reads the configuration file and outputs **kernel_cfg.c**

```
#define STACK_SIZE 2048
#define PRIORITY_DEF 1
char __mip_tsk1_stack[STACK_SIZE];
T_CTSK __mip_tsk1_ctsk;
MIPTSK_DATA __mip_tsk1 = { TASK_MAIN, TA_HLNG|TA_ACT, 1, task_main,
                          PRIORITY_DEF, STACK_SIZE, __mip_tsk1_stack };
/*--- Kernel initialize ---*/
void Application_Initialize(VOID *first_available_memory)
{
    ER errcd;
    /* Initialize object management module */
    errcd = mip_Initialize(first_available_memory, TSZ_KERNEL_POOL_SIZE);
    __mip_tsk1_ctsk.tskatr = __mip_tsk1.tskatr;
    __mip_tsk1_ctsk.exinf = __mip_tsk1.exinf;
    __mip_tsk1_ctsk.task = __mip_tsk1.task;
    __mip_tsk1_ctsk.itskpri = __mip_tsk1.itskpri;
    __mip_tsk1_ctsk.stksz = __mip_tsk1.stksz;
    __mip_tsk1_ctsk.stk = __mip_tsk1.stk;
    errcd = __cre_tsk(__mip_tsk1.tskid, &__mip_tsk1_ctsk);
}
```

kernel_cfg.c : The start-up code for creating **TASK_MAIN**, which has been specified by the static **API** in the configuration file, is created. It has not been mentioned here, but **kernel_cfg.c** outputs the area needed for the kernel to manage the object as well.

2.4 System Initialization Procedure

When the system is reset, first the hardware dependent low-level initialization process is executed. In **Nucleus µiPLUS**, this is started from the symbol, **INT_Initialize** that is provided as the file called **INT_xxx.yyy**. **xxx** differs among the target board, and **yyy** becomes the standard extension of the assembler file, depending upon which assembler is used. Next, the kernel executes **Nucleus PLUS**'s initialization process. This is started from the routine called **INC_Initialize**, provided as **INC.C** file. Next, the kernel calls the initialization routine

called **Application_Initialize**. This is created into a **kernel_cfg.c** file as a default by using the Configurator. This is where the Configurator calls **µPLUS**'s initialization routine, **mIP_Initialize**. When using **µPLUS**, you always have to call **mIP_Initialize** in the top of **Application_Initialize** routine; however, by using the configurator, this process will be automatically created.

The static **API** that has been described in the configuration file is extended within the **Application_Initialize** and is been processed.

The user must create at least one task in the **Application_Initialize** routine. After the Start-up routine is finished, **Nucleus** scheduler activates and the multi-task scheduling starts. The **CPU** will be in the process of interrupt disable until the scheduler is first been called, after the execution of start-up routine from the reset is finished.

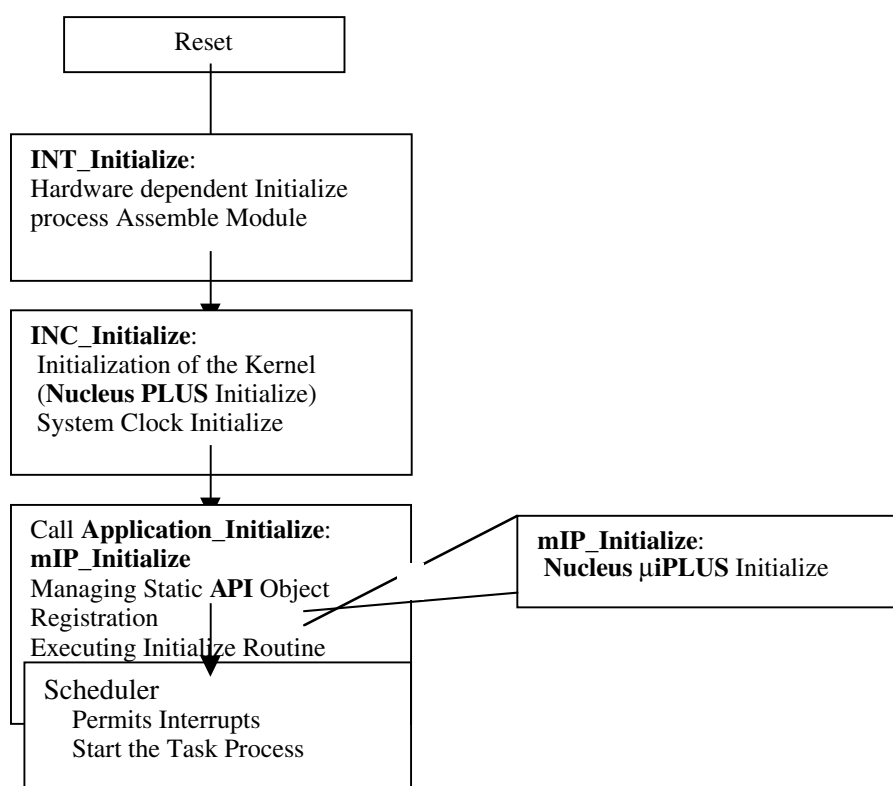


Figure 2.4-1 System Initialization Procedure

2.5 Service call that could be used with **Application_Initialize**

Based on the file output by the configurator, the user can customize the file. Initialization process will be deployed within the **Application_Initialize** routine, but the service call that can be used are only for non-task context and initialization. These service calls will have the same attribute even in the user initialization process, where it has been registered with **ATT_INI**. Initialization service call will be the following service calls where the service call starts with double under score ("_").

__cre_tsk, __cre_mbx, __cre_dtg, __cre_sem, __cre_flg, __cre_mpf,
__cre_mpl, __cre_cyc, __def_tex, __def_inh, __def_exc

Every service calls are the same with the one without having the under score. Also, please do no use these service calls other than initialization process. We cannot assure the behavior when they are used other than for initialization process.

* The service call in this section is specific to **Nucleus µPLUS**.

2.6 Porting Item

Nucleus µPLUS has a number of kernel defines, which the user could easily change. The user could customize the kernel by changing this variable. When these variables are changed, the rebuilt of the kernel

(the recompile of the **Nucleus µiPLUS** library) is needed.

2.6.1 Standard Profile

Nucleus µiPLUS is compliant to **µITRON4.0** Standard Profile. The only service call that can be used is indicated with [S] in the Service Call section in this manual. In order to use the other Extended function, please assign the macro,

EXTEND_PROFILE

when compiling **Nucleus µiPLUS**. By doing so, it enables all of the service call stated in this manual.

2.6.2 Defining Timer Tick

TIC_NUME and **TIC_DENO** are defined in **itron.h** as the Timer Tick definition. These definitions calculate the time as you specify, by combining it with your cyclical interrupt setting of the system hardware.

TIC_NUME	Numerator of Timer Tick cycle (initial value 10)
TIC_DENO	Denominator of Timer Tick cycle (initial value 1)

The initial value is where Timer setting is **10ms**.

Interrupt Cycle of the Hardware Timer (1ms) = TIC_NUME _TIC_DENO

2.6.3 System Memory

Various Middleware for **Nucleus PLUS** can be used in **Nucleus µiPLUS**. These Middleware gets the memory dynamically from **Nucleus PLUS**'s system memory pool. In using **Nucleus PLUS** Middleware (**Nucleus NET**, **Nucleus FILE**, etc.), please define **MIPLUS_USE_SYSMEM** when compiling **Nucleus µiPLUS**, to create this system memory pool. The size of this system memory pool is created with the value in **TSZ_MIP_SYSTEM_MEMORY**, defined by **KERNEL.H**.

3 Functionality of the Kernel

Nucleus μ iPLUS has two functionalities, Real Time function and Multi-task function. When developing an application using **Nucleus μ iPLUS**, system designer must understand the nature of these functions.

3.1 Multi-task Function

When a **JOB** does not use multi-task running on the **CPU**, the **CPU** will be in the idling condition until the input and output is completed. For instance, when you think of such system where a data input, data calculated, and data output has been performed, in order to obtain the best efficiency, calculation should be done while the system is in wait, during the input and output. In order to make this possible, the program for input and output, and the program for calculation must be activated separately. On the **RTOS**, this separate active program is called the **Task**, and the **OS** that allows multiple programming using each Task is called **Multi-task Operating System**. Since **Nucleus μ iPLUS** is a **Multi-task Operating System**, programming using the multi-task function is possible.

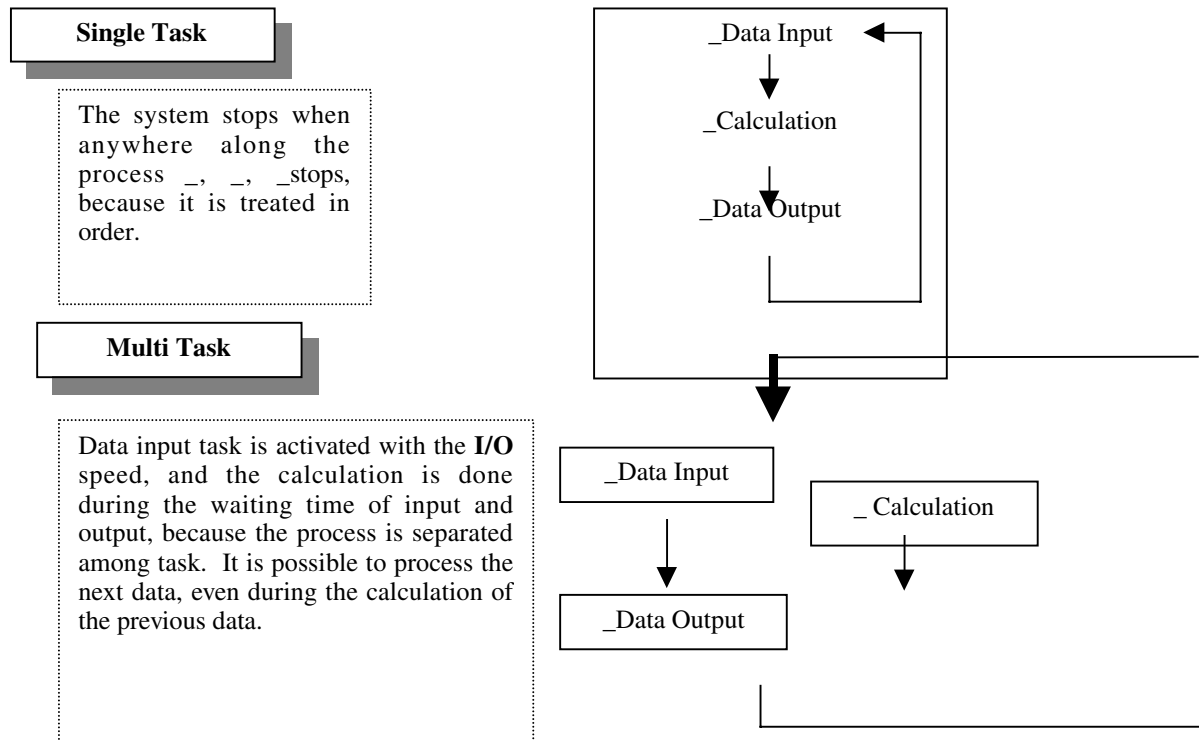


Figure 3.1-1 Benefits of Multi-tasking

3.2 Real Time Function

Real Time function is receiving the event at high speed. High speed means that the process ends within a fixed amount of time. When the Operating System does not have the Real-time function, the time to switch the task and the process time of the Operating System increases as the task within the system increases. **Nucleus μ iPLUS** is a Real Time Operating System, and is designed in a way that the tasks are processed within fixed time, so that it does not influence how the application is constructed. Furthermore, interrupt disable within the kernel is set at minimum, in order to process the interrupts at high speed.

3.3 Task Management

The user could register a certain program as a task to the Operating System. The user activates the task, and can also end the task. The user could also set, in such a way that it would activate right after it is created. The

Operating System constantly watch whether the task is activated, ready to be activated, or been ended. Furthermore, the user sets the priority of the tasks. The task that needs to be activated within a short period of time should be set as high priority task.

Like in these cases, tasks on the Multi-task Operating system have “Status” and “Priority”. Much precise explanation is discussed later.

In order for the user to be able to operate such, the service calls below are available for **Nucleus μ iPLUS**.

cre_tsk	Create Task
acre_tsk	Create Task (automatically assign ID Number)
del_tsk	Delete Task
act_tsk	Activate Task
iact_tsk	Activate Task (from the Interrupt Handler)
can_act	Cancel the Task Activate Request
sta_tsk	Start Task (assigning start code)
ext_tsk	Exit Current Task
ter_tsk	Terminate Task
chg_pri	Change the Priority of the Task
get_pri	Get Task Priority
ref_tsk	Refer to Task status
ref_tst	Refer to Task status (abridged edition)

Table 3.3-1 Task Management Service Call

3.3.1 Task Status

The task that exists above the Multi-task **OS** waits for “Task Status”. The task above **Nucleus μ iPLUS** is either one of the six statuses illustrated below.

(a) **READY**

The task that has been activated will be in this status. With the combination of the task that has the higher priority, the task that is **READY** is not running yet, but it is in the status where it can be activated at any time.

(b) **RUNNING**

The Operating System will activate the **READY** task that has the highest priority. When the task is running and the **OS** chooses it, it is called Dispatch.

(c) **WAITING**

This is the status where the current task has been stopped until a condition of a certain object is ready.

(d) **SUSPENDED**

The task will be in this status, when the service call to suspend a task is issued.

(e) **WAITING-SUSPENDED**

WAITING and **SUSPENDED** are overlapping.

(f) **DORMANT**

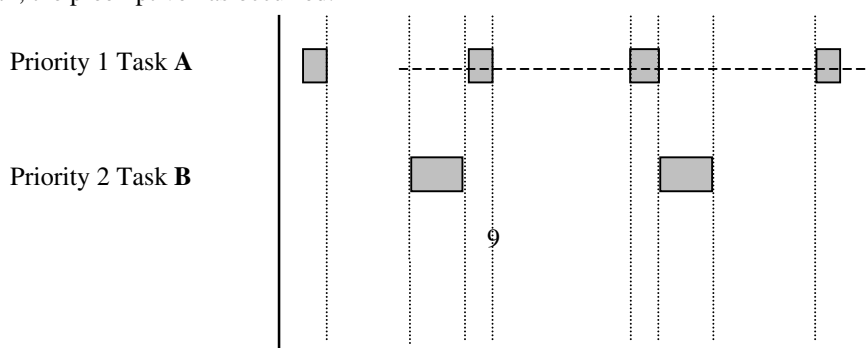
The task is not yet been activated, or the task has finished running.

3.3.2 Scheduling Rule

The user could create number of tasks within the system, and can set the priority to each task. **Nucleus μ iPLUS** is an Operating System based on preemptive priority. This means, the task that has the highest priority will be ready to be activated, and it constantly executes that task.

(Note: “Dispatch Disable” is an exception. It is discussed later in this manual)

If the task with high priority is ready to be activated, it constantly executes this task. When a task that has higher priority than the task that is currently running becomes ready, the task that is running will be forced to loose its activity, and the task that has the higher priority will then be executed. This is what we call, the preemptive has occurred.



Priority 3 Task C

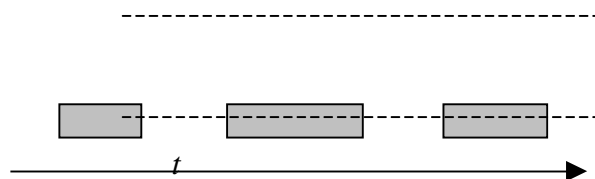


Figure 3.3-2 Scheduling Rule

In the above figure, Task C tries to run with the lowest priority. Task B is a high priority task, where it is activated when an event has been generated. Task A is a much higher priority task, where it is periodically been run. There is no such service call to stop Task C, however, when Task A and B are activated, preemption is generated, and Task C has to wait as **READY** status. When the tasks have the same priority, the task that first switched to ready to be activated will run.

3.3.3 Priority of the Task

Nucleus μ PLUS enables the user to use 1 to 255 task priorities. For your information, μ ITRON4.0 Specification Standard Profile is limited to 1 to 16 task priorities.

3.4 Synchronous Function with Tasks

By issuing a service call that can directly operate the task status, it can activate the task (change from **DORMANT** to **READY**), it can change the task to wait (**WAITING**), or it can change the task to suspended status. By using these service calls, you can let the task wait until a certain process is done, or you can activate the task that was waiting, and synchronize between the tasks.

Nucleus μ PLUS Standard Profile supports the following synchronous function with tasks.

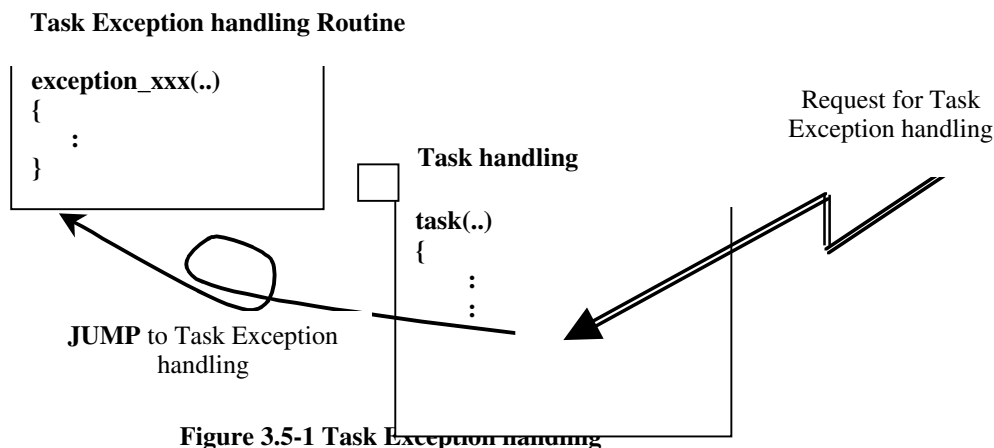
slp_tsk	Sleep Task
tslp_tsk	Sleep Task (with time out)
wup_tsk	Wakeup Task
iwup_tsk	Wakeup Task (from the Interrupt Handler)
can_wup	Canceling of Wakeup Task
rel_wai	Release Waiting Task
irel_wai	Release Waiting Task (from the Interrupt Handler)
sus_tsk	Suspend Task
rsm_tsk	Resume Task
frsm_tsk	Force Resume Task
dly_tsk	Delayed Current Task

Table 3.4-1 Synchronous with Task Service Call

3.5 Task Exception Handling Function

Task Exception handling function is one of the new functions of μ ITRON specification that has been added from μ ITRON 4.0 specifications. Each task can define the task exception handling routine, and this includes enabling and disabling the task exception handling. The task that defined the task exception handling routine can control this as if an interrupt has occurred, when the request for task exception handling happened while

the task exception handling is approved and running. With this function, task-to-task communication can be done in an asynchronous manner.



The kernel manages the Task Exception Handling parameter for each task that has been requested but has not been processed. This is called Withhold Exception Parameter. The Withhold Exception Handling Parameter is set to 0 when the task is activated. The task waits for either Task Exception Process Disable or Authorized. The task that has been just activated is in Task Exception Handling Disabled status. The Task Exception Handling Routine will be activated, only when the following 5 conditions are met.

- Task Exception Handling Routine is defined.
- Task Exception Handling Routine is authorized
- Withhold Exception Handling Parameter is not 0
- Task is in Activating state
- Non Task Context or CPU Exception Handler is not activated

def_tex	Define Task Exception Handling Routine
ras_tex	Request Task Exception Handling
iras_tex	Request Task Exception Handling (from the Interrupt Handler)
dis_tex	Disable Task Exception Handling
ena_tex	Enable Task Exception Handling
sns_tex	Refer to Task Exception Handling Disable status
ref_tex	Refer to Task Exception Handling status

Table 3.5-1 Task Exception Handling Service Call

3.6 Synchronization and Communication Function

Task Synchronization is a function for synchronizing between tasks, where semaphore and event exists when the object that is independent from the task is used. In order to send and receive data between tasks, we have data queue function and mailbox function.

3.6.1 Semaphore

Semaphore is an object that is used for synchronizing and mutual exclusion for a certain resource to use. Semaphore itself has a counter that tells whether it has a resource or not to be corresponded and also tells how many resources are available. When the corresponding resource is been used, it obtains a semaphore by issuing a service call that decreases its resource number by one. When the task finishes using the resource, it returns the resource to the semaphore by issuing a service call that increases its resource by one.

cre_sem	Create Semaphore
acre_sem	Create Semaphore (automatically assign ID Number)
del_sem	Delete Semaphore
sig_sem	Release Semaphore
isig_sem	Release Semaphore (from the Interrupt Handler)
wai_sem	Obtain Semaphore
pol_sem	Obtain Semaphore (Polling)
twai_sem	Obtain Semaphore (with time out)
ref_sem	Refer to Semaphore status

Table 3.6-1 Synchronization and Communication function, Semaphore Service Call

3.6.2 Event Flag

The synchronization of the event flag is achieved by setting the bit inside the event flag **on/off**. Since waiting of an event flag could be done by bit pattern, it is very suited when you want to wait for several events by using only one task. Furthermore, it is best suited for asynchronously telling that the event has only occurred to the other task's independent context.

cre_flg	Create Event Flag
acre_flg	Create Event Flag (automatically assign ID Number)
del_flg	Delete Event Flag
set_flg	Set Event Flag
iset_flg	Set Event Flag (from the Interrupt Handler)
clr_flg	Clear Event Flag
wai_flg	Wait Event Flag
pol_flg	Wait Event Flag (Polling)
twai_flg	Wait Event Flag (with Time Out)
ref_flg	Refer to Event Flag

Table 3.6-2 Synchronization and Communication Event Flag, Service Call

3.6.3 Data Queue

Data queue is an object that synchronizes and communicates by exchanging the word data messages. Data queue has an area called data queue area within itself to store the data, and sending a data to data queue actually means copying data into data area. Furthermore, data queue has a line of tasks that is waiting for a data to be sent and a line of tasks that is waiting for a data to be received.

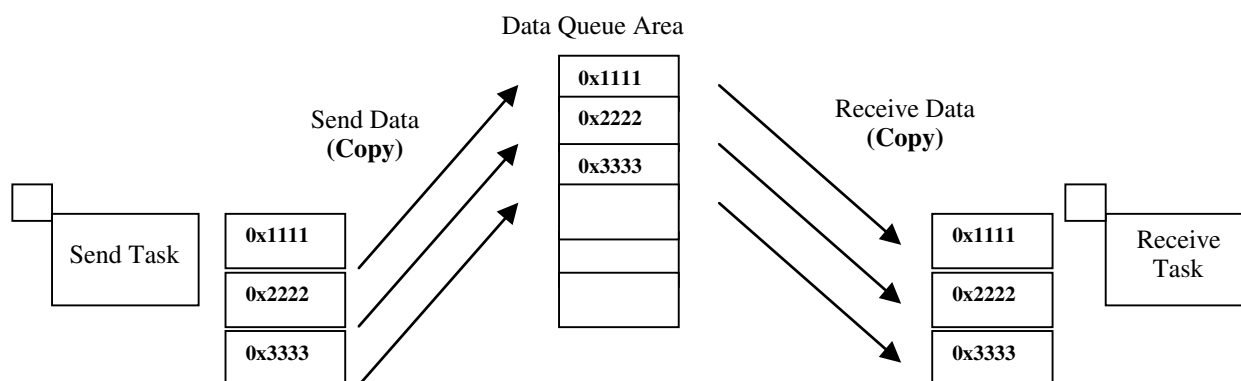


Figure 3.6-3 Communication between tasks by Data Queue

cre_dtq	Create Data Queue
acre_dtq	Create Data Queue (automatically assign ID Number)
del_dtq	Delete Data Queue
snd_dtq	Send to Data Queue
psnd_dtq	Send to Data Queue (Polling)
ipsnd_dtq	Send to Data Queue (from the Interrupt Handler)
tsnd_dtq	Send to Data Queue (with Time Out)
fsnd_dtq	Force Send to Data Queue
ifsnd_dtq	Force Send to Data Queue (from the Interrupt Handler)
rcv_dtq	Receive from Data Queue
prcv_dtq	Receive from Data Queue (Polling)
trcv_dtq	Receive from Data Queue (with Time Out)
ref_dtq	Refer to Data Queue status

Table 3.6-3 Synchronization and Communication function, Data Queue Service Call

3.6.4 Mailbox

Mailbox is an object for task synchronization and communication where message data in the memory is sent and received. In Mailbox, the kernel manages the mailbox by the link list. Because of this reason, the application program must secure an area in the beginning of the message for a kernel to be able to use the link list, before sending the data. This area is called the message header. When the user rewrites the message header area, while the message is still in the link list of the mailbox, there will be no assurance in the outcome. Furthermore, since the kernel rewrites the message header, there will be no assurance in the outcome if you send the same message twice. In mailbox, the pointer of the message header is sent and received, assuming that the area next to the message header exists. Since data length will not be sent and received, the data size must be fixed in the application side or data size information must be included within the data area.

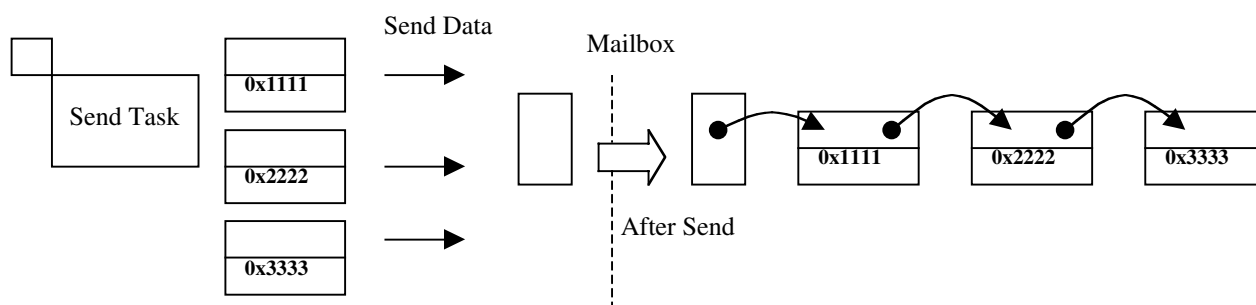


Figure 3.6-4 Send Data to Mailbox

cre_mbx	Create Mailbox
acre_mbx	Create Mailbox (automatically assign ID Number)
del_mbx	Delete Mailbox
snd_mbx	Send to Mailbox
isnd_mbx	Send to Mailbox (from the Interrupt Handler)
rcv_mbx	Receive from Mailbox
prcv_mbx	Receive from Mailbox (polling)
trcv_mbx	Receive from Mailbox (with Time Out)
ref_mbx	Refer to Mailbox status

Table 3.6-4 Synchronization and Communication function Mailbox Service Call

3.7 Memory Pool Management Function

Memory Pool management function is a function that manages the operation of allocating and returning back the memory block against the memory pool, where the memory area used for activating is specified as the memory pool area.

3.7.1 Fixed-sized Memory Pool

Fixed-sized Memory Pool is an object that dynamically manages the memory block that has been fixed. In fixed-sized memory pool function, having a memory area used as fixed-sized memory pool provides services such as allocating and returning the memory block. When the pool has not enough memory and some tasks requests to allocate memory, it creates a line of task that is waiting for allocating the memory block.

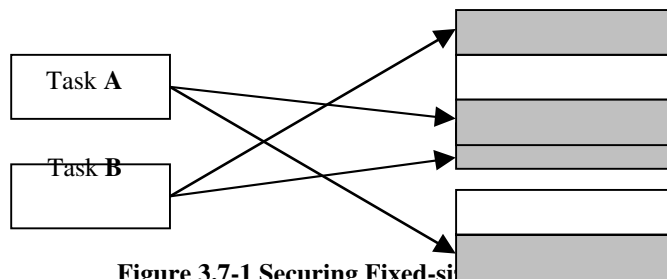


Figure 3.7-1 Securing Fixed-si

cre_mpf	Create Fixed-sized Memory Pool
acre_mpf	Create Fixed-sized Memory Pool (automatically assign ID Number)
del_mpf	Delete Fixed-sized Memory Pool
get_mpf	Get Fixed-sized Memory Block
pget_mpf	Get Fixed-sized Memory Block (Polling)
tget_mpf	Get Fixed-sized Memory Block (with Time Out)
rel_mpf	Return Fixed-sized Memory Block
ref_mpf	Refer to Fixed-sized Memory Block status

Table 3.7-1 Fixed-sized Memory Pool Service Call

Furthermore, Fixed-sized Memory Pool function uses the following macro to seek the necessary area size for creating the pool by setting the block number and size.

SIZE mpfsz = TSZ_MPF(UINT blkcnt, UINT blksz)

blkcnt is the number of pools and **blksz** is the number of bytes for the pool.

3.7.2 Variable-sized Memory Pool

Variable-sized memory pools are objects that dynamically manage variable sized blocks of memory. Variable-sized Memory Pool function has a memory area that is used as Variable-sized Memory Pool, and it provides services such as allocation and deallocation of the memory. Task that allocates memory block from variable-sized memory pool will be in waiting state, until enough memory block is returned from variable-sized memory pool. The task that is in waiting status of variable-sized memory block is connected to the wait queue of variable-sized memory pool. The fragmentation of the pool area occurs, when the allocation and deallocation of different sized memory block takes place because of this service. When the fragmentation occurs, the necessary consecutive block may not be secured, even if there is enough open area. You may want to divide the pool by its usage, or allocate a block that is big in size or does not release so often, in the beginning of the system, because the fragmentation does not get resolved by itself.

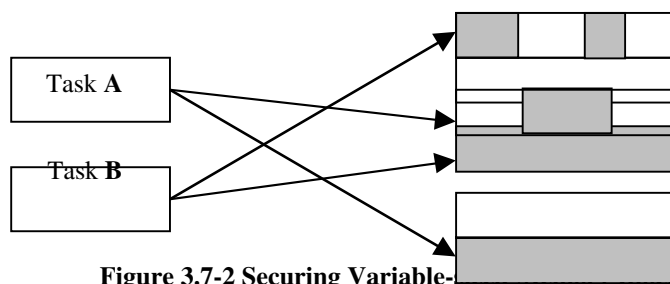


Figure 3.7-2 Securing Variable-

cre_mpl	Create Variable-sized Memory Pool
acre_mpl	Create Variable-sized Memory Pool (automatically assign ID Number)
del_mpl	Delete Variable-sized Memory Pool
get_mpl	Get Variable-sized Memory Block
pget_mpl	Get Variable-sized Memory Block (Polling)
tget_mpl	Get Variable-sized Memory Block (with Time Out)
rel_mpl	Release Variable-sized Memory Block
ref_mpl	Refer to Variable-sized Memory Pool status

Table 3.7-2 Variable-sized Memory Pool Service Call

3.8 Time Management Function

Time Management function is the basic function that operates all of what is time dependent. **Nucleus μ PLUS** has a timer that continuously increment from the time when it is initialized. By using this function as the basis, it provides system time management, cyclical handler, and alarm handler. Cyclical handler, alarm handler, and over run handler are all called time event handler.

3.8.1 System Time Management

In **Nucleus μ PLUS**, the system clock is set to **0** when the system initializes. And then on, it will continuously increment in each interrupt of the hardware timer. Then it provides with system time management where you can refer and set the system time. Even when you change the system clock using the service call for system clock setting, the event and time out process that is set by other service call using relative time is not influenced.

set_tim	Set System Time
get_tim	Refer to System Time

Table 3.8-1 System Time Management Service Call

3.8.2 Cyclic Handler

Cyclic Handler is a time event handler that is executed at a fixed cycle. It is best suited for periodical process. Cyclic Handler allows an operation to start and stop the action.

cre_cyc	Create Cyclic Handler
sta_cyc	Start Cyclic Handler
stp_cyc	Stop Cyclic Handler

Table 3.8-2 Cyclic Handler Service Call

3.9 System Status Management Function

In order to change and refer to all sorts of status in the system that is currently running, various system status management functions can be used.

rot_rdq	Rotate Task Priority Order
irrot_rdq	Rotate Task Priority Order (from the Interrupt Handler)
get_tid	Get the Running Task ID
iget_tid	Get the Running Task ID (from the Interrupt Handler)
loc_cpu	Lock CPU Lock
illoc_cpu	Lock CPU Lock (from the Interrupt Handler)
unl_cpu	Unlock CPU Lock
iunl_cpu	Unlock CPU Lock (form the Interrupt Handler)
dis_dsp	Disable Dispatch
ena_dsp	Enable Dispatch
sns_ctx	Sense Context
sns_loc	Sense CPU Lock status
sns_dsp	Sense Disable Dispatch
sns_dpn	Sense Delayed Dispatch

Table 3.9-1 System Status Management Function, Service Call

3.9.1 Rotate Task Priority Order

According to the scheduling rule, when the tasks with the same priority are ready to be executed, the task will be executed in order of which task becomes ready first. In this case, unless the running task becomes wait, the other task will have to wait in order, until it is executed. By issuing a service call to rotate the task priority order, you can rotate the task waiting line that is been specified. When the specified task is currently running, it will be ready and placed in the end of the line, and the next task that is waiting will be executed.

3.9.2 CPU Lock

When the service call that locks the **CPU** is issued, the **CPU** interrupts will be masked, and will disable interrupts. As an opposition to this, when you relieve the **CPU** lock, the **CPU** will accept the interrupts. Disable and enable interrupts are done against the corresponding **CPU**. Whenever the external has an interrupt controller, you have to control this separately.

3.9.3 Dispatch Disable

When dispatch of the task is been disabled, preemption does not take place, even if the task that has higher priority than the activated task gets ready to be activated. This status is called Dispatch Disable status. When the task issues a system call that will be in Wait status itslef, the task will be in Wait status, and the dispatch occurs. At this point, dispatch disable gets released, but when the control gets back to the task that is in Wait status, dispatch gets disabled again. In this manner, dispatch disable is set towards the task rather than to the system in **Nucleus μ PLUS**.

Note_ In **μ TRON4.0** specification, the rule is that you cannot issue a service call where the task becomes to be in waiting status by itself, during the dispatch disable. The behavior after issuing the service call to become a waiting status during disable dispatch is defined, because of **Nucleus μ PLUS**'s unique specification; therefore, be aware when creating a compatible program on **μ TRON4.0** specification kernel.

3.10 Interrupt Management Function

Nucleus μ PLUS provides a service call that defines the Interrupt Handler.

def_int	Define Interrupt Handler
----------------	---------------------------------

Table 3.10-1 Interrupt Management Service Call

In **Nucleus μ iPLUS**, the switching of the stack and saving of the context before calling the interrupt handler takes place. The context is returned when the dispatch process is been called after the interrupt handler is ended. The stack area for the system is used during the interrupt handler is been activated.

3.11 System Configuration Management Function

It provides a service call that defines the **CPU** Exceptional Handler as the System Configuration Management Function. In **Nucleus μ iPLUS** implementation, there is no distinction between the CPU Exceptional Handler and Interrupt Handler. Also, there are no functional differences between **def_exc** and **def_int**.

There is a service call for referring this configuration information, for the information that the configurator outputs.

def_exc	Define CPU Exceptional Handler
ref_cfg	Refer to Configuration Information

Table 3.11-1 System Configuration Management Service Call

4 Service Call Reference

4.1 Task Management Functions

4.1.1 CRE_TSK Create Task _Static API_S_

How to illustrate Static API

```
CRE_TSK( ID tskid,{ATR tskatr,VP_INT exinf,FP task,RPI itskpri,SIZE
stksz,VP stk });
```

Parameter

ID	tskid	ID number for new task
T_CTSK*	pk_ctsk	Pointer to the packet with task creating information
ATR	tskatr	Task Attribute
VP_INT	exinf	Task Extended Information
FP	task	Task Start Address
PRI	itskpri	Task Priority
SIZE	stksz	Task Stack Size (bytes)
VP	stk	Stack Starting Address

[Function]

The task is activated with high level language interface for setting **TA_HLNG(0x00)**. Since this version does not support the task activation by assembler language interface, this setting can be omitted.

The status of the referent task during the task creation will be Ready, when **TA_ACT(0x02)** is set, and Dormant when it is not.

The memory area from the address set by **stk** to **stksz** byte will be used as stack area for activating the task. When **NULL(=0)** is set to **stk**, the configurator secures the memory area in number of bytes that is set by **stksz**. The task creating information that has been set will be secured in the area prepared by the configurator.

How to illustrate the task in C Language is as follows.

```
void task(VP_INT exing)
{
    main task
    ext_tsk();
}
```

When it is returned from the main routine of the task, it will behave just like when **ext_tsk** is been called.

4.1.2 cre_tsk Create Task

Service Call

ER ercd = cre_tsk(ID tskid,T_CTSK*pk_ctsk);

Parameter

ID	tskid	ID number for new task
T_CTSK*	pk_ctsk	Pointer to packet with task creating information
ATR	tskatr	Task Attribute
VP_INT	exinf	Task Extended Information
FP	task	Task Start Address
PRI	itskpri	Task Priority
SIZE	stksz	Task Stack Size (bytes)
VP	stk	Stack Starting Address

Return Parameter

ER **ercd** Success (**E_OK**) or Error Code

Error Code

E_ID	Invalid ID number (More than the maximum number of tasks or a negative value is set to the Task ID)
E_NOMEM	not enough memory (NULL is set to stk but cannot secure enough stack area)
E_PAR	Parameter error_ pk_ctsk,task,inskpri,stksz , is invalid_
E_RSATR	Reserved Attribute Error
E_OBJ	object status error (referent task has already been registered)
E_CTX	called from Non Task Context
E_NOSPT	Non supported function is set
E_SYS	system error

Function

Tasks that have ID number set by **tskid** are created based on the task creating information set by **pk_ctsk**. **tskatr** is task attribute; **exinf** is extended information handed as a parameter when the task is executed; **task** is task activating address; **itskpri** is activating priority that sets the initialize value of the base priority when the task is executed; **stksz** is the stack area size of the task(in number of bytes); **stk** is the stack area starting address of the task. In **tskatr**, ((**TA_HLNG**) | [**TA_ACT**]) can be set. The task is activated with high level language interface for setting **TA_HLNG(0x00)**. Since this version does not support the task activation by assembler language interface, this setting can be omitted. The status of the referent task during the task creation will be Ready, when **TA_ACT(0x02)** is set, and Dormant when it is not. The memory area from the address set by **stk** to **stksz** byte will be used as stack area for activating the task. When **NULL(=0)** is set to **stk**, the configurator secures the memory area in number of bytes that is set by **stksz**. The task creating information that has been set will be secured in the area prepared by the kernel. Do not destroy the task creating information set by **pk_ctsk** when the task is in valid because the kernel uses this for task functional information.

4.1.3 del_tsk Delete Task

Service Call

ER ercd = del_tsk(ID tskid);

Parameter

ID	tskid	ID Number for the Task to be deleted
-----------	--------------	--------------------------------------

Return Parameter

ER **ercd** Success_**E_OK**_or Error Code



Error Code

E_ID	Invalid ID Number _More than the maximum number of tasks or a negative value is set to the Task ID _
E_NOEX S	Object does not exist (the task was not specified)
E_CTX	Called from Non-task Context
E_OBJ	Invalid object state (specified invoking task's ID or task not in DORMANT state)
E_SYS	System Error

Function

Delete the task that has the **ID** number set by **tskid**.

4.1.4 acre_tsk Create Task (ID number automatically assigned)

Service Call

ER_ID tskid = acre_tsk(T_CTSK*pk_ctsk);

Parameter

T_CTSK*	pk_ctsk	Pointer to the packet with task creating information
ATR	tskatr	Task Attribute
VP_INT	exinf	Task Extended Information
FP	task	Task Start Address
PRI	itskpri	Task Priority
SIZE	stksz	Task Stack Size (bytes)
VP	stk	Stack Starting Address

Return Parameter

ER_ID taskid **ID** number of created task or error code

Error Code

E_NOID	Insufficient ID numbers for tasks
E_NOMEM	Insufficient memory (NULL has been set to stk but can not secure the stack area)
E_PAR	Parameter Error pk_ctsk,task,inskpri,stksz,stk is invalid_
E_CTX	Called from Non-task Context
E_NOSPT	Non supported function is set
E_SYS	System Error

Function

Task is created based on the task creating information set by **pk_ctsk**. **ID** that has not been allocated in the task management area is allocated when creating the **ID** number.
Please refer to **cre_tsk** for further information on **pk_ctsk**.

4.1.5 **act_tsk** **Activate Task_S_**

Service Call

ER **ercd=act_tsk(ID tskid);**

Parameter

ID **tskid** **ID** number of task to be activated

Return Parameter

ER **ercd** Success **E_OK** or Error Code

Error Code

E_ID	Invalid ID Number_More than the maximum number of tasks or a negative value is set to the Task ID_
E_NOEX S	Object does not exist (The task was not specified)
E_QOVR	Queue over flow (overflow of activate request queuing count)
E_CTX	Called from Non-task Context
E_NOID	Referent task could not be found
E_SYS	System Error

Function

Executes the task set by **tskid**. The activating request towards the task is queued when the referent task is not in dormant.

4.1.6 **iact_tsk** Activate Task_Exclusive for Non-task Context__S_

Service Call

ER **ercd**=**iact_tsk**(**ID** **tskid**);

Parameter

ID **tskid** **ID** number of task to be activated

Return Parameter

ER **ercd** Success_**E_OK**_or Error Code

Error Code

E_ID Invalid **ID** Number_More than the maximum number of tasks or a negative value is set to the Task **ID**
E_NOEX Object does not exist (The task was not specified)
E_NOMEM Delayed Execution Buffer is full
E_OBJ Object Status Error (**TSK_SELF** is been set)
E_CTX Called from Non-task Context

Function

Executes the task set by **tskid**. The activating request towards the task is queued when the referent task is not in dormant.

4.1.7 **can_act** Cancel Activation Request _S_

Service Call

ER_UINT actcnt=can_act(ID tskid);

Parameter

ID tskid ID number of canceling activate request

Return Parameter

ER_UINT actcnt Activate Request Count that was queued (>0 or = current count) or error code

Error Code

E_ID	Invalid ID Number_More than the maximum number of tasks or a negative value is set to the Task ID
E_NOEXS	Object does not exist (a task was not specified)
E_CTX	Called from Non-task Context
E_NOID	Referent task could not be found

Function

It cancels the activating request that has been queued against the task that is set by **tskid**, and returns the number of activating requests that has been queued. When **TSK_SELF(=0)** is set to **tskid**, the invoking task will be the referent task.

4.1.8 **sta_tsk** Start a task with a start code

Service Call

ER **errcd** = **sta_tsk**(**ID** **tskid**, **VP_INT** **stacd**);

Parameter

ID **tskid** **ID** number of task to be activated
VP_INT **stacd** Starting code of task

Return Parameter

ER **errcd** Success **E_OK** or Error Code

Error Code

E_ID Invalid **ID** Number_More than the maximum number of tasks or a negative value is set to the Task **ID**
E_NOEX Object does not exist (a task was not specified)
E_CTX Called from Non-task Context
E_OBJ Invalid object state (the task state is not DORMANT)
E_SYS System Error

Function

It activates the task set by **tskid**. It sends the **stacd** value as the parameter, when it activates the task.

4.1.9 **ext_tsk** Terminate the invoking task _S_

Service Call

void ext_tsk();

Parameter

None

Return Parameter

None (This service call will not return when it is successfully completed)

Function

It ends the invoking task. When the activating request towards the invoking task is queued, subtract **1** from the number of queues from the activating request, and change the invoking task to ready status. When the error is found internally, it will return from this service call but the error information will not be returned.

4.1.10 **ter_tsk** Terminate a task_S_

Service Call

ER ercd=ter_tsk(ID tskid);

Parameter

ID tskid ID number of the task to be terminated

Return Parameter

ER ercd Success_**E_OK**_or Error Code

Error Code

E_ID Invalid ID Number_More than the maximum number of tasks or a negative value is set to the Task ID_
E_NOEXS Object does not exist (a task was not specified)
E_ILUSE Service Call is illegal (call for the invoking task)
E_OBJ Invalid object state (the task's state is not DORMANT)
E_CTX Called from Non-task Context
E_SYS System Error

Function

It terminates the task set by **tskid**. When the activating request towards the referent task is being queued, subtract **1** from the number of queues from the activating request, and change the referent task to ready status.

The invoking task cannot be terminated with this service call.

4.1.11 **chg_pri** Change a task's priority_S_

Service Call

ER ercd=chg_pri(ID tskid,PRI tskpri);

Parameter

ID tskid ID number of the referent modified task
PRI tskpri New base priority for modified task

Return Parameter

ER ercd Success_**E_OK**_or Error Code

Error Code

E_ID	invalid ID Number_More than the maximum number of tasks or a negative value is set to the Task ID_
E_NOEXS	object does not exist (a task was not specified)
E_PAR	parameter Error_ tskpri is invalid_
E_ILUSE	service call is illegal (limit priority)
E_OBJ	invalid object state (task's state was not DORMANT)
E_CTX	Called from Non-task Context
E_NOID	Referent task could not be found

Function

It changes the base priority of the task set by **tskid** to the value set by **tskpri**. Accordingly, the current priority of the task changes as well.
When **TSK_SELF(=0)** is set to **tskid**, the invoking task becomes the referent task. Also, when **TPRI_INI(=0)** is set to **tskpri**, the base priority of the referent task is changed to priority during task execution.

4.1.12 **get_pri** Get task priority_S_

Service Call

ER **ercd**=**get_pri**(**ID** **tskid**,**PRI*****p_tskpri**);

Parameter

ID **tskid** **ID** number of the refered task

Return Parameter

ER **ercd** Success_**E_OK**_or Error Code
PRI **tskpri** Current priority of the task

Error Code

E_ID Invalid **ID** Number_More than the maximum number of tasks or a negative value is set to the Task **ID**
E_NOEXS Object does not exist (a task was not specified)
E_OBJ Invalid object state (the task's state is not DORMANT)
E_CTX Called from Non-task Context
E_NOID Referent task could not be found

Function

It refers the current priority of the task set by **tskid**, and stores it in the area set by **p_tskpri**. When **TSK_SELF(=0)** is set to **tskid**, the invoking task becomes the referent task.

4.1.13 ref_tsk Reference a task's state

Service Call

ER ercd = ref_tsk(ID tskid,T_RTsk*pk_rtsk);

Parameter

ID tskid ID number of the status referred task
T_RTsk* pk_rtsk Pointer to packet returning task information

Return Parameter

ER ercd Success_E_OK_or Error Code

Contents of **pk_rtsk** (Type: T_RTsk*)

STAT tskstat Task state. Return one of the following values.

TTS_RUN 0x01 RUNNING state
TTS_RDY 0x02 READY state
TTS_WAI 0x04 WAITING state
TTS_SUS 0x08 SUSPENDED state
TTS_WAS 0x0C WAITING-SUSPENDED state
TTS_DMT 0x10 DORMANT state

PRI tskpri Current priority of task

PRI tsbpri Base priority of task

Both of these will be same value in the current version.

STAT tsawait Reason for waiting. The value is 0, when the referent task is not in WAITING state. One of the following values is returned, when the referent task is in WAITING STATE.

TTW_SLP 0x0001 Sleeping
TTW_DLY 0x0002 Delayed
TTW_SEM 0x0004 Waiting for a semaphore
TTW_FLG 0x0008 Waiting for an event flag
TTW_SDTQ 0x0010 Waiting for sending to a data queue
TTW_RDTQ 0x0020 Waiting for receiving from a data queue
TTW_MBX 0x0040 Waiting for receiving from a mail box
TTW_MPF 0x2000 Waiting for getting a fixed-sized memory block
TTW_MPL 0x4000 Waiting for getting a variable-sized memory block

ID wobjid ID Number of Object, which task is waiting. The value is 0, when the task is not in WAITING state.

TMO lefttmo Time left to Time Out. The value is 0, when the referent task is not in WAITING state.

UINT actcnt Number of activate request queuing

UINT wupcnt Number of Wake-up request queuing

UINT suscnt Number of Suspend request queuing

Error Code

E_ID Invalid ID Number_More than the maximum number of tasks or a negative value is set to the Task ID_

E_NOEXS Object does not exist (a task was not specified)

E_PAR Parameter Error (NULL has been set to **pk_rtsk** *)

E_CTX Called from Non-task Context

E_NOID Referent task could not be found

Function

It refers the status regarding the task set by **tskid**, and stores it in the area set by **pk_rtsk**. When **TSK_SELF(=0)** is set to **tskid**, the invoking task becomes the referent task. For **tskpri** and **tsbpri**, even when the referent task is in Dormant, it returns the value that is currently set. Also, when the task is not in Wait status, **tsawait** and **wobjid** will be 0.



4.1.14 ref tst Reference a task's state (minimal function)

Service Call

```
ER    ercd = ref_tst(ID_tskid,T_RTST*pk_rtst);
```

Parameter

ID	tskid	ID number of the referent task with information reference
----	-------	---

T_RTST* **pk_rtst** Pointer to the packet returning task information

Return Parameter

ER **ercd** Success **E OK** or Error Code

Contents of **pk_rtst** (Type: **T_RTST***)

STAT **taskstat** Task state. Return one of the following values.

TTS RUN 0x01 RUNNING state

TTS RDY 0x02 READY state

TTS WAI 0x04 WAITING state

TTS SUS 0x08 SUSPENDED state

TTS WAS 0x0C WAITING-SUSPENDED state

TTS DMT 0x10 DORMANT state

STAT **tskwait** Reason for waiting. The value is **0**, when the referent task is not in WAITING state. One of the following values is returned, when the referent task is in WAITING STATE.

TTW SLP 0x0001 Sleeping

TTW DLY 0x0002 Delayed

TTW SEM 0x0004	Waiting for a semaphore
----------------	-------------------------

TTW FLG 0x0008 Waiting for an event flag

TTW_SDTQ 0x0010 Waiting for sending to a data queue

TTW_RDTQ 0x0020 Waiting for receiving from a data queue

TTW_MBX 0x0040 Waiting for receiving from a mail box

TTW_MPF 0x2000 Waiting for getting a fixed-sized memory block

TTW MPL 0x4000 Waiting for getting a variable-sized

memory block

Error Code

E_ID Invalid **ID** Number_More than the maximum number of tasks or a negative value is set to the Task **ID**

E NOEXS Object does not exist (a task was not specified)

E PAR Parameter Error **NULL** has been set to **pk rtst ***

CTX	Called from Non-task Context
E	

E NOID	Referent tasks could not be found
---------------	-----------------------------------

Function

It refers the status regarding the task set by **tskid** and stores it in the area set by **pk_rtst**. When **TSK_SELF(=0)** is set to **tskid**, the invoking task becomes the referent task. This service call is an abridged version of **ref_tsk()**.

4.2 Task dependent synchronization function

4.2.1 **slp_tsk** Put task to sleep_S_

Service Call

ER ercd=slp_tsk();

Parameter

None

Return Parameter

ER ercd Success_**E_OK**_or Error Code

Error Code

E_RLWAI WAITING state was terminated (**rel_wai** was called while in WAITING state)

E_CTX Called from Non-task Context

E_SYS System Error

Function

It changes the invoking task in to task sleep. However, when the sleep requests towards the invoking task is in queue, subtract **1** from the sleep request queue and continue running without changing the invoking task to Wait status.

Supplement

E_OK is returned when wait status is released by **wup_tsk** been accepted, and **E_RLWAI** is returned by wait status is released by **rel_wai** been accepted, after becoming in wait status because of this service.

4.2.2 tslp_tsk Put task to sleep with a timeout_S_

Service Call

ER ercd=tslp_tsk(TMO tmout);

Parameter

TMO **tmout** Time out

Return Parameter

ER **ercd** Success_**E_OK**_or Error Code

Error Code

E_PAR Parameter Error_**tmout** is invalid_
E_RLWAI WAITING state was terminated (**rel_wai** was called while in WAITING state)
E_TMOUT Polling failed or timeout occurred
E_CTX Called from Non-task Context
E_SYS System Error

Function

It changes the invoking task to sleep. However, when the sleep requests towards the invoking task is in queue, subtract **1** from the sleep request queue and continue running without changing the invoking task to Wait status.

For **tmout**, adding to the positive value of timeout, **TMO_POL(=0)** and **TMO_FEVR(=-1)** can be set.

4.2.3 wup_tsk Wakeup a task_S_

Service Call

ER ercd=wup_tsk(ID tskid);

Parameter

ID **tskid** **ID** number of the referent wakeup task

Return Parameter

ER **ercd** Success_ **E_OK** or Error Code

Error Code

E_ID	Invalid ID Number_More than the maximum number of tasks or a negative value is set to the Task ID
E_NOEXS	Object does not exist (a task was not specified)
E_OBJ	Invalid object state (the task state is not DORMANT)
E_QOVR	Queue over flow (overflow of activate request count)
E_CTX	Called from Non-task Context
E_NOID	Referent task could not be found
E_SYS	System Error

Function

Release Wait from Sleep Wait status on the task set by **tskid**. For the task that is release from Wait status, return **E_OK** as the returned value of the service call that is in the Wait status. When the referent task is not in Sleep Wait status, the sleep request towards the task will be in queue. When the number of queue exceeds the maximum number, return **E_QOVR**. When **TSK_SELF(=0)** is set to **tskid**, the invoking task becomes the referent task.

4.2.4 iwup_tsk Wakeup a Task (exclusive to Non-task context)_S_

Service Call

ER ercd=iwup_tsk(ID tskid);

Parameter

ID **tskid** **ID** number of the referent wakeup task

Return Parameter

ER **ercd** Success_**E_OK**_or Error Code

Error Code

E_ID	Invalid ID Number_More than the maximum number of tasks or a negative value is set to the Task ID
E_NOEXS	Object does not exist (a task was not specified)
E_OBJ	Invalid object state (the task state is not DORMANT)
E_NOMEM	Delayed Execution Buffer is full
E_CTX	Called from the task context

Function

Release Wait from Sleep Wait status on the task set by **tskid**. For the task that is release from Wait status, return **E_OK** as the returned value of the service call that is in the Wait status. When the referent task is not in Sleep Wait status, the sleep request towards the task will be in queue. When the number of queue exceeds the maximum number, return **E_QOVR**. With this service call, when **TSK_SELF(=0)** is set to **tskid**, it will be **E_ID** error.

4.2.5 **can_wup** Cancel wakeup request **_S_**

Service Call

ER_UINT wupcnt=can_wup(ID tskid);

Parameter

ID tskid **ID** number of the referent wakeup cancel requesting task

Return Parameter

ER_UINT wupcnt Wakeup request count (**>0** or **=0**) or error code

Error Code

E_ID	Invalid ID Number_More than the maximum number of tasks or a negative value is set to the Task ID_
E_NOEXS	Object does not exist (a task was not specified)
E_OBJ	Invalid object state (the task state is not DORMANT)
E_CTX	Called from Non-task Context
E_NOID	Referent task could not be found
E_SYS	System Error

Function

Cancels the Sleep request that is queued towards the task set by **tskid**, and returns the number of Sleep request that has been queued.

When **TSK_SELF(=0)** is set to **tskid**, the invoking task becomes the referent task.

4.2.6 rel_wai Release task from WAITING state_S_

Service Call

ER ercd=rel_wai(ID tskid);

Parameter

ID **tskid** **ID** number of the referent task to be forcefully released from wait task

Return Parameter

ER **ercd** Success_**E_OK**_or Error Code

Error Code

E_ID Invalid **ID** Number_More than the maximum number of tasks or a Negative value is set to the Task **ID**
E_NOEXS Object does not exist (a task was not specified)
E_OBJ Invalid object state (the task state is not DORMANT)
E_CTX Called from Non-task Context

Function

When the task set by **tskid** is in Wait status, it forcefully releases the Wait status. In other words, when the referent task is in Wait status, it will be Ready status; when it is in Double Wait status, it will be Suspend status. For the task that has been release from Wait status using this service call, **E_RLWAI** is returned as the returned value of the service call that is in Wait status.

4.2.7 **irel_wai** Release task from WAITING state_exclusive to Non-task context__S_

Service Call

ER **ercd=irel_wai(ID tskid);**

Parameter

ID **tskid** **ID** number of the referent task to be forcefully released from wait task

Return Parameter

ER **ercd** Success_**E_OK**_or Error Code

Error Code

E_ID	Invalid ID Number_More than the maximum number of tasks or a negative value is set to the Task ID
E_NOEXS	Object does not exist (task was not specified)
E_OBJ	Invalid object state (set TSK_SELF to subjected task)
E_NOMEM	Delayed Execution Buffer is full
E_CTX	Called from task context

Function

When the task set by **tskid** is in Wait status, it forcefully releases the Wait status. In other words, when the referent task is in Wait status, it will be Ready status; when it is in Double Wait status, it will be Suspend status. For the task that has been release from Wait status using this service call, **E_RLWAI** is returned as the returned value of the service call that is in Wait status.

4.2.8 **sus_tsk** Suspend a task_S_

Service Call

ER **ercd=sus_tsk(ID tskid);**

Parameter

ID **tskid** **ID** number of the referent transition task

Return Parameter

ER **ercd** Success_**E_OK**_or Error Code

Error Code

E_ID	Invalid ID Number_More than the maximum number of tasks or a negative value is set to the Task ID
E_NOEXS	Object does not exist (task was not specified)
E_OBJ	Invalid object state (task state is not DORMANT)
E_QOVR	Queue over flow (overflow of activate request count)
E_CTX	Called from Non-task Context
E_NOID	Referent task could not be found
E_SYS	System Error

Function

Interrupts the task execution by suspending a task set by **tskid**. When **TSK_SELF(=0)** is set to **tskid**, the invoking task becomes the referent task.

4.2.9 **rsm_tsk** Resume a suspended task_S_

Service Call

ER **ercd=rsm_tsk(ID tskid);**

Parameter

ID **tskid** **ID** number of the referent task to be resumed

Return Parameter

ER **ercd** Success_**E_OK**_or Error Code

Error Code

E_ID	Invalid ID Number_More than the maximum number of tasks or a negative value is set to the Task ID _
E_NOEXS	Object does not exist (task was not specified)
E_OBJ	Invalid object state (task state is not DORMANT)
E_CTX	Called from Non-task Context
E_CTX	System Error

Function

It releases the suspend status of the task set by **tskid**, and resume the task execution. When the number of suspend request nest becomes **0**, it will be Ready if the referent task is in Suspend, and it will be in Wait if the referent task is in Double Wait Status.

4.2.10 frsm_tsk Forcefully resume a suspended task_S_

Service Call

ER ercd=frsm_tsk(ID tskid);

Parameter

ID tskid **ID** number of the refeent task to be resumed

Return Parameter

ER **ercd** Success_**E_OK**_or Error Code

Error Code

E_ID Invalid **ID** Number_More than the maximum number of tasks or a negative value is set to the Task **ID**
E_NOEXS Object does not exist (task was not specified)
E_OBJ Invalid object state (task state is not DORMANT)
E_CTX Called from Non-task Context
E_SYS System Error

Function

It releases the suspend status of the task set by **tskid**, and resume the task execution. This service call sets the suspend request nest to **0**, and it will be Ready if the referent task is in Suspend, and it will be in Wait if the referent task is in Double Wait Status.

4.2.11 dly_tsk Delay the invoking task_S_

Service Call

ER ercd=dly_tsk(RELTIM dlytim);

Parameter

RELTIM dlytim Amount of time to delay the invoking task (relative time)

Return Parameter

ER **ercd** Success_**E_OK**_or Error Code

Error Code

E_PAR Parameter Error (**dlytim** is invalid)
E_RLWAI Forcibly released from WAITING state (**rel_wai** was invoked while in WAITING state)
E_CTX Called from Non-task Context
E_NOEXS Object not created (invoking task is not registered)
E_SYS System Error

Function

Change the invoking task to Time Elapse Wait status, and during the time set by **dlytim**, it pauses the execution.

4.3 Task exception handling functions

4.3.1 DEF_TEX Define a task exception handling routine (static API)_S_

How to illustrate static API

```
DEF_TEX(ID tskid,{ATR texatr,FP texrtn});
```

Parameter

ID	tskid	ID number of the task to be defined
ATR	texatr	Attribute of task exception handling routine
FP	texrtn	Start address of task exception handling routine

Function

Based on the Task Exception Handling routine Define information, define the task exception handling routine to the task set by **tskid**. **Tskid** will automatically allocate the non-supported integer value parameter, and **texatr** will set the preprocessor constant number parameter. (**TA_HLNG**) could be set in **texatr**. Task Exception Handling routine is executed in high language interface for setting **TA_HLNG**. Since this version does not support assembler language interface in executing the Task Exception Handling routine, this setting could be omitted.

The referent task must be created by **CRE_TSK** in the configuration file, illustrated before **DEF_TEX**.

How to illustrate Task Exception handling routine in C language is as follows.

```
void texrtn(TEXTPTN texptn, VP_INT exinf)
{
    task exception handling routine itself
}
```

4.3.2 **def_tex** Define a task exception handling routine

Service Call

ER **ercd**=**def_tex**(**ID** **tskid**,**T_DTEX*****pk_dtex**);

Parameter

ID	tskid	ID number of the task to be defined
T_DTEX*	pk_dtex	Pointer to the packet containing the task exception handling routine
Contents of pk_dtex (T_DTEX type)		
ATR	texatr	Attribute of task exception handling routine
FP	texrtn	Start address of task exception handling routine

Return Parameter

ER **ercd** Success **E_OK** or Error Code

Error Code

E_ID	Invalid ID Number_More than the maximum number of tasks or a negative value is set to the Task ID
E_NOEXS	Object does not exist (a task was not specified)
E_RSATR	Reserved attribute (texatr was invalid or could not be used)
E_PAR	Parameter Error_ pk_dtex , texrtn is invalid_
E_CTX	Called from Non-task Context
E_NOID	Referent task could not be found
E_SYS	System Error
E_NOSPT	Nonsupported Function is set

Error Code

Based on the Task Exception Handling routine Define information set by **pk_dtex**, define the task exception handling routine to the task set by **tskid**. When **NULL** is set to **pk_dtex**, it releases the Task Exception Handling routine that has been set, and it will be in the status where Task Exception Handling routine is not yet set. When **TSK_SELF(=0)** is set to **tskid**, the invoking task becomes the referent task. (**TA_HLNG**) could be set in **texatr**. Task Exception Handling routine is executed in high language interface for setting **TA_HLNG**. Since this version does not support assembler language interface in executing the Task Exception Handling routine, this setting could be omitted.

How to illustrate Task Exception handling routine in C language is as follows.

```
void texrtn(TEXTPTN texptn, VP_INT exinf)
{
    task exception handling routine itself
}
```

4.3.3 **ras_tex** Raise task exception handling_S_

Service Call

ER ercd=**ras_tex**(**ID** tskid,**TEXPTN** raspn);

Parameter

ID	tskid	ID number of the referent demanding task
TEXPTN	raspn	Task exception code for task exception handling

Return Parameter

ER	ercd	Success_ E_OK _or Error Code
-----------	-------------	-------------------------------------

Error Code

E_ID	Invalid ID Number_More than the maximum number of tasks or a negative value is set to the Task ID
E_NOEXS	Object does not exist (a task was not specified)
E_OBJ	Invalid object state (the task state is not DORMANT or task exception handling routine was not defined)
E_CTX	Called from Non-task Context
E_PAR	Parameter Error (0 is set to raspn)
E_NOID	Referent task could not be found
E_CTX	System Error

Function

Request the Task Exception Handling by Task Exception factor set by **raspn**, towards the task set by **tskid**. The reserved exception factor will be updated in the logical addition per bit of the **raspn** value that has been set.
When **TSK_SELF(=0)** is set to **tskid**, the invoking task becomes the referent task.

4.3.4 iras_tex Raise task exception handling (exclusive to Non-task context)_S_

Service Call

ER ercd=iras_tex(**ID** tskid,TEXPTN rasptn);

Parameter

ID	tskid	ID number of the referent demanding task
TEXPTN	rasptn	Task exception code for task exception handling

Return Parameter

ER	ercd	Success_ E_OK _or Error Code
-----------	-------------	-------------------------------------

Error Code

E_ID	Invalid ID Number_More than the maximum number of tasks or a negative value is set to the Task ID
E_NOEXS	Object does not exist (task was not specified)
E_NOMEM	Delayed Execution Buffer is full
E_CTX	Called from task context
E_OBJ	Object Status Error (TSK_SELF is set to ID)
E_PAR	Parameter Error (0 is set to rasptn)

Function

Request the Task Exception Handling by Task Exception factor set by **rasptn**, towards the task set by **tskid**. The reserved exception factor will be updated in the logical addition per bit of the **rasptn** value that has been set.

When **TSK_SELF**(=0) is set to **tskid**, **E_ID** error is returned.

4.3.5 **dis_tex** **Disable task exception handling_S_**

Service Call

ER **ercd=dis_tex();**

Parameter

None

Return Parameter

ER **ercd** Success_**E_OK**_or Error Code

Error Code

E_OBJ	Invalid object state (task exception handling routine was not defined)
E_CTX	Called from Non-task Context
E_NOID	Cannot find the invoking task
E_ID	Invalid ID Number (ID number registering information of the invoking task is invalid)
E_NOEXS	Object is not created (invoking task is not registered)

Function

Changes the invoking task to Task exception handling disable status.

4.3.6 **ena_tex** Enable task exception handling_S_

Service Call

ER **ercd=ena_tex();**

Parameter

None

Return Parameter

ER **ercd** Success_**E_OK**_or Error Code

Error Code

E_OBJ	Invalid object state (task exception handling routine was not defined)
E_CTX	Called from Non-task Context
E_NOID	Cannot find the invoking task
E_ID	Invalid ID Number (ID number registering information of the invoking task is invalid)
E_NOEXS	Object not created (invoking task is not registered)

Function

Changes the invoking task to Enable Task Excepton Handling status.
When all of the attributes to execute Task Exception Handling Routine match by this service call, Task Exception Handling Routine will execute.

4.3.7 **sns_tex** Sense task exception handling status **_S_**

Service Call

BOOL state=sns_tex();

Parameter

None

Return Parameter

BOOL state Task exception disabled state

TRUE is returned when the Task Exception Handling Routine is disabled on the activating task. **TRUE** is also returned even when the activating task does not exist.

Function

Returns the Task Exception Handling Disabled status on the activating task (when called from Task Context, it will be the invoking task).

4.3.8 ref_tex Reference state of task exception handling _S_

Service Call

ER ercd=ref_tex(**ID** taskid, **T_RTEX** *pk_rtex);

Parameter

ID	tskid	ID number of the referent task for information reference
T_RTEX*	pk_rtex	Pointer to packet for returning state of task exception handling
STAT	texstat	State of task exception handling
TEXPTN	pndptn	Pending exception code

Return Parameter

ER	ercd	Success E_OK or Error Code
-----------	-------------	-----------------------------------

Error Code

E_ID	Invalid ID Number_More than the maximum number of tasks or a negative value is set to the Task ID
E_NOEXS	object does not exist (a task was not specified)
E_PAR	Parameter Error_ pk_rtex is invalid
E_OBJ	Invalid object state (the task state is not DORMANT or task exception Handling routine was not defined)
E_CTX	Called from Non-task Context
E_NOID	Referent task could not be found

Function

Refers the information related on Task Exception Handling for the task set by **tskid**, and stores it in the area set by **pk_rtex**.

One of the following values is returned from **texstat**.

TTEX_ENA	0x00	Task Exception Handling Enable Status
TTEX_DIS	0x01	Task Exception Handling Disable Status

Reserved Exception Parameter is returned to **pndptn**. When the Exception Handling request that has not been processed exists, **pndptn** returns **0**.

When **TSK_SELF(=0)** is set to **tskid**, the invoking task becomes the referent task. **E_OBJ** is returned when the referent task is in DORMANT.

4.4 Synchronization and Communication Functions - Semaphore

4.4.1.1 CRE_SEM Create Semaphore (Static API)_S_

How to illustrate static API

```
CRE_SEM(ID semid,{ATR sematr,UINT isemcnt,UINT maxsem});
```

Parameter

ID	semid	ID number of the created semaphore
ATR	sematr	semaphore attribute
UINT	isemcnt	initial value of resource count of semaphore
UINT	maxsem	maximum resource count of semaphore

Function

Creates the semaphore that has the **ID** number set by **semid**, based on the semaphore creating information that has been set. **semid** sets the automatic allocation non support integer value parameter, and **sematr** sets the preprocessor constant parameter.

(**TA_TFIFO** | **TA_TPRI**) can be set in **sematr**. Semaphore waiting queue will be in order of **FIFO**, when **TA_TFIFO(=0x00)** is set, and it will be in order of task priority when **TA_TPRI(=0x01)** is set.

4.4.2 cre_sem Create Semaphore

Service Call

ER ercd=cre_sem(ID semid,T_CSEM*pk_csem);

Parameter

ID	semid	ID number of the created semaphore
T_CSEM* pk_csem		Pointer to packet with semaphore creation information
ATR	sematr	semaphore attribute
	TA_TFIFO	(Semaphore according to the order of FIFO)
	TA_TPRI	(Semaphore according to the order of priority)
UINT	isemcnt	initial value of resource count of semaphore
UINT	maxsem	maximum resource count of semaphore

Return Parameter

ER **ercd** Success_**E_OK**_or Error Code

Error Code

E_ID	Invalid ID Number_More than the maximum number of semaphore or a negative value is set to the Task ID
E_RSATR	reserve attribute (sematr is invalid or unusable)
E_PAR	Parameter Error (pk_csem , isemcnt , maxsem are invalid)
E_OBJ	object state error (semaphore is already registered)
E_CTX	Called from Non-task Context

Function

Creates the semaphore that has the **ID** number set by **semid**, based on the semaphore creating information set by **pk_csem**. (**TA_TFIFO** || **TATPRI**) can be set in **sematr**. (**TA_TFIFO** | **TA_TPRI**) can be set in **sematr**. Semaphore waiting queue will be in order of **FIFO**, when **TA_TFIFO(=0x00)** is set, and it will be in order of task priority when **TA_TPRI(=0x01)** is set. Do not destroy the task creating information set by **pk_csem** when the task is in valid because the kernel uses this for task functional information.

4.4.3 **acre_sem** Create Semaphore (ID number automatically assignment)

Service Call

ER_ID semid=acre_sem(T_CSEM*pk_csem);

Parameter

ID	semid	ID number of the created semaphore
T_CSEM*	pk_csem	Pointer to packet with semaphore creation information
ATR	sematr	semaphore attribute
	TA_TFIFO	(Semaphore according to the order of FIFO)
	TA_TPRI	(Semaphore according to the order of priority)
UINT	isemcnt	initial value of resource count of semaphore
UINT	maxsem	maximum resource count of semaphore

Return Parameter

ER_ID	semid	ID number (positive number) of the created semaphore or error code
--------------	--------------	---

Error Code

E_NOID	Invalid ID Number (there is no Semaphore ID assignable)
E_RSATR	Reserve attribute (sematr is invalid or unusable)
E_PAR	Parameter Error (pk_csem , isemcnt , maxsem are invalid)
E_CTX	Called from Non-task Context

Function

Creates the semaphore based on the semaphore creating information set by **pk_csem**. When creating the **ID** number, the **ID** that has not been allocated in the semaphore management area is used. Please refer to **cre_sem** for **pk_csem**.

4.4.4 **del_sem** Delete Semaphore

Service Call

ER **ercd** = **del_sem**(**ID** **semid**);

Parameter

ID **semid** **ID** number of the semaphore to be deleted

Return Parameter

ER **ercd** Success_**E_OK**_or Error Code

Error Code

E_ID invalid **ID** (**semid** is invalid or unusable)

E_NOEX S uncreated object (referent semaphore is unregistered)

E_CTX Called from Non-task Context

Function

Deletes the semaphore set by **semid**.

4.4.5 sig_sem Release Semaphore_S_

Service Call

ER ercd=sig_sem(**ID** semid);

Parameter

ID **semid** **ID** number of the semaphore receiving released resource

Return Parameter

ER **ercd** Success_**E_OK**_or Error Code

Error Code

E_ID	invalid ID (semid is invalid or unusable.)
E_NOEXS	uncreated object (referent semaphore is unregistered)
E_QOVR	queue overflow (release will exceed maximum resource count)
E_CTX	Called from Non-task Context
E_OBJ	Object is invalid

Function

Return **1** resource against the semaphore set by **semid**.

4.4.6 isig_sem Release Semaphore_exclusive to Non-task context__S_

Service Call

ER ercd=isig_sem(ID semid);

Parameter

ID **semid** **ID** number of the semaphore receiving released resource

Return Parameter

ER **ercd** Success_**E_OK**_or Error Code

Error Code

E_ID invalid **ID** (semid is invalid or unusable.)
E_NOEXS uncreated object (referent semaphore is unregistered)
E_NOMEM Delayed Execution Buffer is full
E_CTX Called from task context

Function

Return **1** resource against the semaphore set by **semid**.
 This service call is only used for Non Task Context, therefore cannot detect **E_QOVR** (Queue overflow) because of delayed execution.

4.4.7 wai_sem Get Semaphore_S_

Service Call

ER ercd=wai_sem(ID semid);

Parameter

ID **semid** **ID** number for the semaphore getting resources

Return Parameter

ER **ercd** Success_**E_OK**_or Error Code

Error Code

E_ID	invalid ID (semid is invalid or unusable.)
E_NOEXS	uncreated object (referent semaphore is unregistered)
E_RLWAI	forced release of suspend state (accept rel_wai while wait)
E_DLT	delete wait object (semaphore is deleted while wait)
E_CTX	Called from Non-task Context
E_OBJ	Object is invalid

Function

Get **1** resource from the semaphore set by **semid**.

When there is no resource available to get, it changes to the semaphore resource get wait status. When other task is already in wait queue, it hangs the invoking task up to the given position of the wait queue based on the semaphore attribute.

4.4.8 **pol_sem** Get Semaphore (polling)_S_

Service Call

ER **ercd**=**pol_sem**(**ID** **semid**);

Parameter

ID **semid** **ID** number for the semaphore getting resources

Return Parameter

ER **ercd** Success_**E_OK**_or Error Code

Error Code

E_ID	invalid ID (semid is invalid or unusable.)
E_NOEXS	uncreated object (referent semaphore is unregistered)
E_TMOUT	fail of poling or timeout
E_CTX	Called from Non-task Context
E_OBJ	Object is invalid
E_DLT	Delete object

Function

Get **1** resource from the semaphore set by **semid**.

Return **E_TMOUT** instead of changing to wait status, when there is no resource available to get.

4.4.9 twai_sem Get Semaphore (timeout)_S_

Service Call

ER ercd=**twai_sem**(**ID** semid,**TMO** tmout);

Parameter

ID	semid	ID number for the semaphore getting resources
TMO	tmout	specify timeout

Return Parameter

ER	ercd	Success E_OK or Error Code
-----------	-------------	-----------------------------------

Error Code

E_ID	invalid ID (semid is invalid or unusable.)
E_NOEXS	uncreated object (referent semaphore is unregistered)
E_PAR	Parameter Error tmout is invalid
E_RLWAI	forced release of suspend state (accept rel_wai while wait)
E_TMOUT	fail of poling or timeout
E_DLT	delete wait object (semaphore is deleted while wait)
E_CTX	Called from Non-task Context
E_OBJ	Object is invalid

Function

Get **1** resource from the semaphore set by **semid**.

When there is no resource available to get, it changes to the semaphore resource get wait status. When other task is already in wait queue, it hangs the invoking task up to the given position of the wait queue based on the semaphore attribute.

In **tmout**, **TMO_POL(=0)** and **TMO_FEVR(=-1)** can be set, addition to the positive value timeout time. When **TMO_POL** is set, it will act in the same way as **pol_sem**. Also, when **TMO_FEVR** is set, it will at in the same way as **wai_sem**.

4.4.10 **ref_sem** Refer Semaphore State

Service Call

ER **ercd** = **ref_sem**(**ID** **semid**,**T_RSEM*****pk_rsem**);

Parameter

ID **semid** **ID** number of the referent semaphore with information reference
T_RSEM* **pk_rsem** pointer to packet that will contain the semaphore information

Return Parameter

ER **ercd** Success **E_OK** or Error Code
Content of **T_RSEM*** **pk_rsem**
 ID **wtskid** **ID** of the head task in the wait queue of the semaphore
 UINT **semcnt** present resource count

Error Code

E_ID invalid **ID** (**semid** is invalid or unusable.)
E_NOEXS uncreated object (referent semaphore is unregistered)
E_PAR Parameter Error **NULL** has been set to **pk_rsem ***
E_CTX Called from Non-task Context

Function

Refer to the semaphore status set by **semid**, and stores it in the area set by **pk_rsem**.

4.5 Synchronization and Communication Functions - Eventflag

4.5.1 CRE_FLG Create an eventflag (Static API)_S_

How to illustrate static API

```
CRE_FLG(ID flgid,{ATR flagatr,FLGPtn iflgptn});
```

Parameter

ID	flgid	ID number of specified eventflag that will be created
ATR	flagatr	attribute of eventflag
FLGPtn	iflgptn	initial value of bit pattern for eventflag

Function

Creates the eventflag that has the **ID** number set by **flgid**, based on the eventflag creating information that has been set. **flgid** sets the automatic allocation non support integer value parameter, and **flagatr** sets the preprocessor constant parameter.

((TA_TFIFO || TA_TPRI) |(TA_WSGl | TA_WMUL) | [TA_CLR]) can be set in **flagatr**. Eventflag waiting queue will be in order of **FIFO** when **TA_TFIFO(=0x00)** is set, and it will be in order of task priority when **TA_TPRI(=0x01)** is set. When **TA_WSGl(=0x00)** is set, it does not allow for more than one task to be in Wait status simultaneously with one Eventflag. When **TA_WMUL(=0x02)** is set, it allows more than one task to be in Wait status simultaneously. When **TA_CLR(=0x04)** is set, it clears all bits in the Eventflag bit pattern, in order to release the task from Wait status, when they are in Event wait status.

4.5.2 cre_flg Create an eventflag

Service Call

ER ercd=cre_flg(**ID** flgid,T_CFLG*pk_cflg);

Parameter

ID	flgid	ID number of specified eventflag to be created (except acre_flg)
T_CFLG*	pk_cflg	Pointer to a packet of creation information for eventflag.
	ATR	attribute of eventflag
	TA_TFIFO	eventflag according to the order of FIFO
	TA_TPRIO	eventflag according to the order of priority
	TA_WSGL	only allow one task to wait
	TA_WMUL	more than one task can wait on this event
FLGPNTN	iflgptn	initial value of bit pattern for eventflag

Return Parameter

ER **ercd** Success_**E_OK** or Error Code

Error Code

E_ID	invalid ID (flgid is invalid or unusable)
E_RSATR	reserve attribute (flgatr is invalid or unusable)
E_PAR	Parameter Error (pk_cflg,iflgptn is invalid)
E_OBJ	object state error (eventflag is already registered)
E_CTX	Called from Non-task Context

Function

Creates the eventflag that has the **ID** number set by flgid, based on the eventflag creating information that has been set. **flgid** sets the automatic allocation non support integer value parameter, and **flgatr** sets the preprocessor constant parameter.

((**TA_TFIFO** || **TA_TPRI**) | (**TA_WSGL** | **TA_WMUL**) | [**TA_CLR**]) can be set in **flgatr**. Eventflag waiting queue will be in order of **FIFO**, when **TA_TFIFO**(=0x00) is set, and it will be in order of task priority when **TA_TPRI**(=0x01) is set. When **TA_WSGL**(=0x00) is set, it does not allow for more than one task to be in Wait status simultaneously with one Eventflag. When **TA_WMUL**(=0x02) is set, it allows more than one task to be in Wait status simultaneously. When **TA_CLR**(=0x04) is set, it clears all bits in the Eventflag bit pattern, in order to release the task from Wait status, when they are in Event wait status.

Do not destroy the task creating information set by **pk_cflg** when the task is in valid because the kernel uses this for task functional information.

4.5.3 acre_flg Create an eventflag (ID number automatically assignment)

Service Call

ER_ID flgid=acre_flg(T_CFLG*pk_cflg);

Parameter

T_CFLG* pk_cflg Pointer to a packet of creation information for eventflag.
ATR flgatr attribute of eventflag
TA_TFIFO eventflag according to the order of **FIFO**
TA_TPRIO eventflag according to the order of priority
TA_WSGL only allow one task to wait
TA_WMUL more than one task can wait on this event
FLGPTN iflgptn initial value of bit pattern for eventflag

Return Parameter

ER_ID flgid **ID** number of created eventflag (positive value) or error code

Error Code

E_NOID lack of **ID** number (there is no eventflag **ID** assignable)
E_RSATR reserve attribute (**flgatr** is invalid or unusable)
E_PAR Parameter Error **pk_cflg,iflgptn** is invalid_
E_CTX Called from Non-task Context

Function

Creates the Eventflag based on the Eventflag creating information set by **pk_cflg**. When creating the **ID** number, the **ID** that has not been allocated in the Eventflag management area is used. Please refer to **cre_flg** for **pk_cflg**.

4.5.4 **del_flg** Delete eventflag

Service Call

ER **ercd** = **del_flg**(**ID flgid**);

Parameter

ID **flgid** **ID** number of eventflag to be deleted

Return Parameter

ER **ercd** Success_**E_OK**_or Error Code

Error Code

E_ID invalid **ID** (**flgid** is invalid or unusable.)
E_NOEX S uncreated object (referent eventflag is unregistered)
E_CTX Called from Non-task Context

Function

Deletes the Eventflag set by **flgid**.

4.5.5 **set_flg** Set Eventflag_S_

Service Call

ER **ercd**=**set_flg**(**ID** **flgid**,**FLGPTN** **setptn**);

Parameter

ID	flgid	ID number of eventflag to be set
FLGPTN	setptn	bit pattern to set

Return Parameter

ER	ercd	Success E_OK or Error Code
-----------	-------------	-----------------------------------

Error Code

E_ID	invalid ID (flgid is invalid or unusable.)
E_NOEXS	uncreated object (referent eventflag is unregistered)
E_CTX	Called from Non-task Context
E_PAR	0 is set to setptn

Function

Sets the bit set by **setptn**, to the Eventflag set by **flgid**. When all of the bits are **0**, return **E_PAR**.

When all of the task wait criteria that the Eventflag have are fulfilled, after the bit pattern of the referent Eventflag is updated, release the appropriate task from Wait status. Also, during this process, if **TA_CLR** is set to the referent Eventflag attribute, clear all bits in the bit pattern of the Eventflag, and end the service call. When **TA_WMUL** is set to the Eventflag attribute and **TA_CLR** is not been set, there might be a possibility where more than one task get released from Wait status with calling this service call once.

4.5.6 **iset_flg** Set eventflag_exclusive to Non-task context__S_

Service Call

ER ercd=iset_flg(**ID flgid**,**FLGPTN setptn**);

Parameter

ID	flgid	ID number of eventflag to be set
FLGPTN	setptn	bit pattern to set

Return Parameter

ER	ercd	Success_ E_OK _or Error Code
-----------	-------------	-------------------------------------

Error Code

E_ID	invalid ID (flgid is invalid or unusable.)
E_NOEXS	uncreated object (referent eventflag is unregistered)
E_NOMEM	Delayed Execution Buffer is full
E_CTX	Called from task context
E_PAR	0 is set to setptn

Function

Sets the bit set by **setptn**, for the Eventflag set by **flgid**. Please refer to **set_flg** for more detail.

4.5.7 **clr_flg** Clear Eventflag_S_

Service Call

ER **ercd=clr_flg(ID flgid,FLGPTN clrptn);**

Parameter

ID	flgid	ID number of eventflag to be cleared
FLGPTN	clrptn	bit pattern to clear (bit-wise turn over)

Return Parameter

ER	ercd	Success_ E_OK _or Error Code
-----------	-------------	-------------------------------------

Error Code

E_ID	invalid ID (flgid is invalid or unusable.)
E_NOEXS	uncreated object (referent eventflag is unregistered)
E_CTX	Called from Non-task Context
E_PAR	1 is set to all bits on clrptn

Function

Clears the bit that is **0**, which corresponds to **clrptn**, for the Eventflag set by **flgid**. When all of the bits are **1**, **E_PAR** is returned.

4.5.8 **wai_flg** Wait for Eventflag_S_

Service Call

ER **ercd**=**wai_flg**(**ID flgid**,**FLGPTN waiptn**,**MODE wfmode**,**FLGPTN*p_flgptn**);

Parameter

ID	flgid	ID number of the eventflag to be waited upon
FLGPTN	waiptn	wait bit pattern
MODE	wfmode	wait mode

Return Parameter

ER	ercd	Success E_OK or Error Code
FLGPTN	p_flgptn	bit pattern of releasing a task from waiting

Error Code

E_ID	invalid ID (flgid is invalid or unusable.)
E_NOEXS	uncreated object (referent eventflag is unregistered)
E_PAR	Parameter Error waiptn , wfmode , p_flgptn , tmout are invalid_
E_ILUSE	invalid use of a service call (The eventflag has TA_WSGL specified and there is already a waiting task)
E_RLWAI	forced release from WAITING state (accept rel_wai while WAITING state)
E_DLT	delete wait object (eventflag is deleted while in WAITING state)
E_CTX	Called from Non-task Context

Function

Wait until the Wait Release status criteria set by **waiptn** and **wfmode** are fulfilled, on the bit pattern of the Eventflag set by **flgid**. When this service call is called, and the Wait Release status criteria of the referent Eventflag bit pattern is fulfilled, it ends the service call rather than shifting the invoking task to Wait status.

(**TWF_ANDW** || **TWF_ORW**) can be set to **wfmode**. Wait status will be released when **TWF_ANDW**(=0x00) is set, and when all bits set by **waiptn** are set, and also when **TWF_ORW**(=0x01) is set, and when either of the bits set by **waiptn** are set.

4.5.9 **pol_flg** Wait for Eventflag (Polling)_S_

Service Call

ER **ercd**=**pol_flg**(**ID flgid**,**FLGPTN waiptn**,**MODE wfmode**,**FLGPTN*p_flgptn**);

Parameter

ID	flgid	ID number of the eventflag to be waited upon
FLGPTN	waiptn	wait bit pattern
MODE	wfmode	wait mode

Return Parameter

ER	ercd	Success E_OK or Error Code
FLGPTN	p_flgptn	bit pattern of releasing a task from waiting

Error Code

E_ID	invalid ID (flgid is invalid or unusable.)
E_NOEXS	uncreated object (referent eventflag is unregistered)
E_PAR	Parameter Error waiptn , wfmode , p_flgptn , tmout are invalid
E_ILUSE	invalid use of a service call (The eventflag has TA_WSGL specified and there is already a waiting task)
E_TMOUT	polling failed or timeout
E_CTX	Called from Non-task Context

Function

Wait until Wait Release Criteria set by **waiptn** and **wfmode** on the bit pattern of the Eventflag set by **flgid** are fulfilled. This service call ends the service call rather than the invoking task becoming in Wait status. Please refer to **wai_flg** for the details on **wfmode**.

4.5.10 twai_flg Wait for Eventflag with Timeout _S_

Service Call

ER ercd=twai_flg(ID lgid,FLGPTN waiptn,MODE wfmode,FLGPTN*p_flgptn,TMO tmout);

Parameter

ID	flgid	ID number of the eventflag to be waited upon
FLGPTN	waiptn	wait bit pattern
MODE	wfmode	wait mode
TMO	tmout	specify timeout

Return Parameter

ER	ercd	Success E_OK or Error Code
FLGPTN	p_flgptn	bit pattern of releasing a task from waiting

Error Code

E_ID	invalid ID (flgid is invalid or unusable.)
E_NOEXS	uncreated object (referent eventflag is unregistered)
E_PAR	Parameter Error_ waiptn,wfmode,p_flgptn,tmout are invalid_
E_ILUSE	invalid use of a service call (The eventflag has TA_WSGL specified and there is already a waiting task)
E_RLWAI	forced release from WAITING state (accept rel_wai while WAITING state)
E_TMOUT	polling failed or timeout
E_DLT	delete wait object (eventflag is deleted while in WAITING state)
E_CTX	Called from Non-task Context

Function

Wait until Wait Release Criteria set by **waiptn** and **wfmode** on the bit pattern of the Eventflag set by **flgid** are fulfilled. When this service call is called, and the bit pattern of the referent Eventflag does not meet the Wait release criteria, end the service call instead of making the invoking task to Wait status. Please refer to **wai_flg** for the detail on wfmode.

In **tmout**, **TMO_POL(=0)** and **TMO_FEVR(=-1)** can be set, addition to the positive value timeout time. When **TMO_POL** is set, it will act in the same way as **pol_flg**. Also, when **TMO_FEVR** is set, it will at in the same way as **wai_flg**.

4.5.11 **ref_flg** **Reference Eventflag Status**

Service Call

ER **ercd** = **ref_flg**(**ID flgid**,**T_RFLG *pk_rflg**);

Parameter

ID **flgid** **ID** number of eventflag whose state is being referenced
T_RFLG* pk_rflg pointer to a packet containing the state of the eventflag

Return Parameter

ER **ercd** Success **E_OK** or Error Code
 Content of **T_RFLG* pk_rflg**
ID **wtskid** **ID** number of head task eventflag's wait queue
FLGPTN flgptn eventflag's current bit pattern

Error Code

E_ID invalid **ID** (**flgid** is invalid or unusable.)
E_NOEXS uncreated object (referent eventflag is unregistered)
E_PAR Parameter Error **NULL** has been set to **pk_rflg***
E_CTX Called from Non-task Context

Function

Refers the information related to Eventflag set by **flgid**, and stores it in the area set by **pk_rflg**. When there is no task in the referent Eventflag queue, **TSK_NONE(=0)** will be set to **wtskid**.

4.6 Synchronization and Communication Functions – Data queue

4.6.1 CRE_DTQ Create data queue (Static API)_S_

How to illustrate static API

```
CRE_DTQ(ID dtqid,{ATR dtqatr,UINT dtqcnt,VP dtq});
```

Parameter

ID	dtqid	ID number of data queue to create
ATR	dtqatr	data queue attribute
UINT	dtqcnt	capacity of data queue (# of data elements)
VP	dtq	the head address of data queue domain

Function

Creates the data queue that has the **ID** number set by **dtqid**, based on the data queue creating information that has been set. **dtqid** sets the automatic allocation non support integer value parameter, and **dtqatr** sets the preprocessor constant parameter.

(**TA_TFIFO** || **TA_TPRI**) can be set in **dtqatr**. Data queue waiting queue will be in order of **FIFO** when **TA_TFIFO(=0x00)** is set, and it will be in order of task priority when **TA_TPRI(=0x01)** is set. When **NULL** is set to **dtq**, the configurator secures the necessary area.

4.6.2 cre_dtq Create data queue

Service Call

ER ercd=cre_dtq(ID dtqid,T_CDTQ*pk_cdtq);

Parameter

ID	dtqid	ID number of data queue to create
T_CDTQ*	pk_cdtq	pointer to packet of data queue creation information
Contents of pk_cdtq_Type: T_CDTQ_		
ATR	dtqatr	data queue attribute
UINT	dtqcnt	capacity of data queue (# of data elements)
VP	dtq	the head address of data queue domain

Return Parameter

ER ercd Success_ **E_OK**_or Error Code

Error Code

E_ID	invalid ID (dtqid is invalid or unusable)
E_NOMEM	lack of memory (can not allocate data queue domain)
E_RSATR	reserved attribute (dtqatr is invalid or unusable)
E_PAR	Parameter Error_ pk_cdtq,dtqcnt,dtq are invalid_
E_OBJ	object state error (referent data queue is already registered)
E_CTX	Called from Non-task Context

Function

Creates the data queue that has the **ID** number set by **dtqid**, based on the data queue creating information that has been set. (**TA_TFIFO** || **TA_TPRI**) can be set in **dtqatr**. Data queue waiting queue will be in order of **FIFO** when **TA_TFIFO(=0x00)** is set, and it will be in order of task priority when **TA_TPRI(=0x01)** is set. When **NULL** is set to **dtq**, the kernel secures the necessary area. Do not destroy the task creating information set by **pk_cdtq** when the task is in valid because the kernel uses this for task functional information.

4.6.3 acre_dtq Create data queue (ID number automatically assignment)

Service Call

ER_ID dtqid=acre_dtq(T_CDTQ*pk_cdtq);

Parameter

T_CDTQ* **pk_cdtq** pointer to packet of data queue creation information
 Contents of **pk_cdtq_Type: T_CDTQ_**
ATR dtqatr data queue attribute
UINT dtqcnt capacity of data queue (# of data elements)
VP dtq the head address of data queue domain

Return Parameter

ER_ID dtqid **ID** number of created data queue (positive value) or error code

Error Code

E_NOID lack of **ID** number (no assignable data queue **ID**)
E_NOMEM lack of memory (can not allocate data queue domain)
E_RSATR reserved attribute (**dtqatr** is invalid or unusable)
E_PAR Parameter Error **pk_cdtq, dtqcnt, dtq** are invalid_
E_CTX Called from Non-task Context

Function

Creates the Data queue based on the Data queue creating information set by **pk_cdtq**. When creating the **ID** number, the **ID** that has not been allocated in the Data queue management area is used. Please refer to **cre_dtq** for **pk_cdtq**.

4.6.4 **del_dtq** Delete data queue

Service Call

ER **ercd** = **del_dtq**(**ID dtqid**);

Parameter

ID **dtqid** **ID** number of data queue to delete

Return Parameter

ER **ercd** Success **E_OK** or Error Code

Error Code

E_ID invalid **ID** (**dtqid** is invalid or unusable)
E_NOEX S uncreated object (referent data queue is unregistered)
E_CTX Called from Non-task Context

Function

Deletes the Data queue set by **dtqid**. When the kernel secures the Data queue area, that area will be released.

4.6.5 **snd_dtq** send to data queue **S_**

Service Call

ER ercd=snd_dtq(**ID** dtqid,VP_INT data);

Parameter

ID	dtqid	ID number of the data queue to be sent
VP_INT	data	data element to send to the data queue

Return Parameter

ER	ercd	Success E_OK or Error Code
-----------	-------------	-----------------------------------

Error Code

E_ID	invalid ID (dtqid is invalid or unusable)
E_NOEXS	uncreated object (referent data queue is unregistered)
E_RLWAI	forced release the WAITING state (accept rel_wai while waiting)
E_DLT	object of wait was deleted (data queue deleted while task was in the WAITING state)
E_CTX	Called from Non-task Context

Function

Sends the data set by **data** to the Data queue set by **dtqid**. When there is a task that is waiting to receive data on the same referent Data queue, it gives the data that is to be sent to the first task in the queue, and release that task from Wait. When the task is not waiting for receiving the data, data that is to be sent is placed in the last of the queue. When there is no open space in the Data queue area, the invoking task is put in the Send Wait queue, and the status changes to Send Wait to the Data queue.

4.6.6 psnd_dtq send to data queue (polling)_S_

Service Call

ER ercd=psnd_dtq(**ID** dtqid,VP_INT data);

Parameter

ID	dtqid	ID number of the data queue to be sent
VP_INT	data	data element to send to the data queue

Return Parameter

ER	ercd	Success_ E_OK _or Error Code
-----------	-------------	-------------------------------------

Error Code

E_ID	invalid ID (dtqid is invalid or unusable)
E_NOEXS	uncreated object (referent data queue is unregistered)
E_TMOUT	fail in polling or timeout
E_CTX	Called from Non-task Context
E_DLT	Delete the Waiting Object

Function

Sends the data set by **data** to the Data queue set by **dtqid**. When there is a task that is waiting to receive data on the same referent Data queue, it gives the data that is to be sent to the first task in the queue, and release that task from Wait. When the task is not waiting for receiving the data, data that is to be sent is placed in the last of the queue. When there is no open space in the Data queue area, **E_TMOUT** is returned.

4.6.7 ipsnd_dtq send to data queue (exclusive to Non-task context)_S_

Service Call

ER ercd=ipsnd_dtq(**ID dtqid**,**VP_INT data**);

Parameter

ID	dtqid	ID number of the data queue to send
VP_INT	data	data element to send to the data queue

Return Parameter

ER	ercd	Success E_OK or Error Code
-----------	-------------	-----------------------------------

Error Code

E_ID	invalid ID (dtqid is invalid or unusable)
E_NOEXS	uncreated object (referent Data queue is not registered)
E_TMOUT	fail in polling or timeout
E_CTX	Called from task context
E_NOMEM	Delayed execution buffer file

Function

Sends the data set by **data** to the Data queue set by **dtqid**. When there is a task that is waiting to receive data on the same referent Data queue, it gives the data that is to be sent to the first task in the queue, and release that task from Wait. When the task is not waiting for receiving the data, data that is to be sent is placed in the last of the queue.

4.6.8 **tsnd_dtq** send to data queue (timeout)_S_

Service Call

ER **ercd**=**tsnd_dtq**(**ID** **dtqid**,**VP_INT** **data**,**TMO** **tmout**);

Parameter

ID	dtqid	ID number of the data queue to send
VP_INT	data	data element to send to the data queue
TMO	tmout	timeout

Return Parameter

ER	ercd	Success_ E_OK _or Error Code
-----------	-------------	-------------------------------------

Error Code

E_ID	invalid ID (dtqid is invalid or unusable)
E_NOEXS	uncreated object (referent eventflag is unregistered)
E_PAR	Parameter Error_ tmout is invalid_
E_RLWAI	forced release the WAITING state (accept rel_wai while waiting)
E_TMOUT	fail in polling or timeout
E_DLT	object of wait was deleted (data queue deleted while task was in the WAITING state)
E_CTX	Called from Non-task Context

Function

Sends the data set by **data** to the Data queue set by **dtqid**. When there is a task that is waiting to receive data on the same referent Data queue, it gives the data that is to be sent to the first task in the queue, and release that task from Wait. When the task is not waiting for receiving the data, data that is to be sent is placed in the last of the queue. When there is no open space in the Data queue area, the invoking task is placed in the end of the Send Wait queue, and changes the status to Send Wait of the Data queue.

In **tmout**, **TMO_POL**(=0) and **TMO_FEVR**(=-1) can be set, addition to the positive value timeout time. When **TMO_POL** is set, it will act in the same way as **psnd_dtq**. Also, when **TMO_FEVR** is set, it will at in the same way as **snd_dtq**.

4.6.9 fsnd_dtq forced send to data queue_S_

Service Call

ER ercd=fsnd_dtq(ID dtqid,VP_INT data);

Parameter

ID	dtqid	ID number of the data queue to send
VP_INT	data	data element to send to the data queue

Return Parameter

ER	ercd	Success_ E_OK _or Error Code
-----------	-------------	-------------------------------------

Error Code

E_ID	invalid ID (dtqid is invalid or unusable)
E_NOEXS	uncreated object (referent data queue is unregistered)
E_ILUSE	illegal use of service call (the capacity of the data queue is 0 (zero))
E_CTX	Called from Non-task Context
E_DLT	Delete Wait Object (referent Data queue is deleted during Wait status)

Function

Sends the data set by **data** to the Data queue set by **dtqid**. When there is a task that is waiting to receive data on the same referent Data queue, it gives the data that is to be sent to the first task in the queue, and release that task from Wait. When the task is not waiting for receiving the data, data that is to be sent is placed in the last of the queue. When there is no open space in the Data queue area, it deletes the first data in the Data queue, and secures the necessary Data queue area. This service call cannot be used when the Data queue area is **0** to the Data queue.

4.6.10 ifsnd_dtq forced send to data queue_S_

Service Call

ER ercd=ifsnd_dtq(ID dtqid,VP_INT data);

Parameter

ID	dtqid	ID number of the data queue to send
VP_INT	data	data element to send to the data queue

Return Parameter

ER	ercd	Success_ E_OK _or Error Code
-----------	-------------	-------------------------------------

Error Code

E_ID	invalid ID (dtqid is invalid or unusable)
E_NOEXS	uncreated object (referent data queue is unregistered)
E_ILUSE	illegal use of service call (the capacity of the data queue is 0 (zero))
E_CTX	Called from task context
E_NOMEM	Delayed Execution Buffer File

Function

Sends the data set by **data** to the Data queue set by **dtqid**. When there is a task that is waiting to receive data on the same referent Data queue, it gives the data that is to be sent to the first task in the queue, and release that task from Wait. When the task is not waiting for receiving the data, data that is to be sent is placed in the last of the queue. When there is no open space in the Data queue area, it deletes the first data in the Data queue, and secures the necessary Data queue area. This service call cannot be used when the Data queue area is **0** to the Data queue.

4.6.11 rcv_dtq receive data element from data queue_S_

Service Call

ER ercd=rcv_dtq(ID dtqid,VP_INT*p_data);

Parameter

ID dtqid ID number of data queue to receive

Return Parameter

ER ercd Success_**E_OK**_or Error Code
VP_INT p_data data element received from data queue

Error Code

E_ID invalid **ID (dtqid)** is invalid or unusable)
E_NOEXS uncreated object (referent data queue is unregistered)
E_PAR Parameter Error (**p_data** or **tmout** are invalid)
E_RLWAI release by force from waiting (accept **rel_wai** while WAITING state)
E_DLT delete a waiting object (deleted data queue while waiting)
E_CTX Called from Non-task Context

Function

Receives the data from Data queue set by **dtqid**, and secures it in the area set by **p_data**. When the data is in the referent Data queue, the first data is taken. When there is a task waiting to be sent by Data queue, the first task in the Send Wait queue gets the data that is trying to send. When there is no data and no task waiting to send, the invoking task is placed in the Receive Wait queue, and changes its status to Receive Wait from Data queue.

4.6.12 prcv_dtq receive data element from data queue (polling)_S_

Service Call

ER ercd=prcv_dtq(**ID** dtqid,VP_INT*p_data);

Parameter

ID **dtqid** **ID** number of data queue to receive

Return Parameter

ER **ercd** Success_**E_OK**_or Error Code
VP_INT **p_data** data element received from data queue

Error Code

E_ID invalid **ID** (**dtqid** is invalid or unusable)
E_NOEXS uncreated object (referent data queue is unregistered)
E_PAR Parameter Error (**p_data** or **tmout** are invalid)
E_TMOUT fail in polling or timeout
E_CTX Called from Non-task Context
E_DLT Delete Waiting Object

Function

Receives the data from Data queue set by **dtqid**, and secures it in the area set by **p_data**. When the data is in the referent Data queue, the first data is taken. When there is a task waiting to be sent by Data queue, the first task in the Send Wait queue gets the data that is trying to send. When there is no data and no task waiting to send, **E_TMOUT** is returned.

4.6.13 trcv_dtq receive data element from data queue (timeout)_S_

Service Call

ER ercd=trcv_dtq(**ID** dtqid,VP_INT*p_data,TMO tmout);

Parameter

ID	dtqid	ID number of data queue to receive
TMO	tmout	timeout

Return Parameter

ER	ercd	Success_ E_OK or Error Code
VP_INT	p_data	data element received from data queue

Error Code

E_ID	invalid ID (dtqid is invalid or unusable)
E_NOEXS	uncreated object (referent data queue is unregistered)
E_PAR	Parameter Error_ p_data,tmout are invalid_
E_RLWAI	release by force from waiting (accept rel_wai while WAITING state)
E_TMOUT	fail in polling or timeout
E_DLT	delete a waiting object (deleted data queue while waiting)
E_CTX	Called from Non-task Context

Function

Receives the data from Data queue set by **dtqid**, and secures it in the area set by **p_data**. When the data is in the referent Data queue, the first data is taken. When there is a task waiting to be sent by Data queue, the first task in the Send Wait queue gets the data that is trying to send. When there is no data and no task waiting to send, the invoking task is placed in the Receive Wait queue, and changes its status to Receive Wait from Data queue.

In **tmout**, **TMO_POL(=0)** and **TMO_FEVR(=-1)** can be set, addition to the positive value timeout time. When **TMO_POL** is set, it will act in the same way as **prcv_dtq**. Also, when **TMO_FEVR** is set, it will at in the same way as **rcv_dtq**.

4.6.14 **ref_dtq** reference the state of the data queue

Service Call

ER **ercd** = **ref_dtq**(**ID dtqid**,**T_RDTQ*pk_rdtq**);

Parameter

ID **dtqid** **ID** number of referent data queue information
T_RDTQ* pk_rdtq Pointer to location of packet to receive data queue state info

Return Parameter

ER **ercd** Success_ **E_OK** _or Error Code
 Contents of **pk_rdtq** (Type: **T_RDTQ**)
ID **stskid** **ID** number of first (head) task in send wait queue
ID **rtskid** **ID** number of first (head) task in receive wait queue
UINT **sdtqcnt** standardized capacity of the data queue

Error Code

E_ID invalid **ID** (**dtqid** is invalid or unusable)
E_NOEXS uncreated object (referent data queue is unregistered)
E_PAR Parameter Error_ **NULL** has been set to **pk_rdtq*_**
E_CTX Called from Non-task Context

Function

Refers the information related to Data queue set by **dtqid**, and stores it in the area set by **pk_rdtq**.
 When there is no task in the referent Data queue Waiting queue, **TSK_NONE(=0)** will be set to **stskid**.
 When there is no task in the referent Data queue Receive queue, **TSK_NONE(=0)** is set to **rtskid**.

4.7 Synchronization and Communication Functions – Mailbox

4.7.1 CRE_MBX create mailbox (static API)_S_

How to illustrate static API

```
CRE_MBX(ID mbxid,{ATR mbxatr,PRI maxmpri,VP mbrihd});
```

Parameter

ID	mbxid	ID number mailbox to create
ATR	mbxatr	mailbox attribute
PRI	maxmpri	maximum message priority
VP	mbrihd	address for priority-based message queue header

Function

Creates the mailbox that has the **ID** number set by **mbxid**, based on the mailbox creating information that has been set. **mbxid** sets the automatic allocation non support integer value parameter, and **mbxatr** sets the preprocessor constant parameter. **maxmpri** and **mbrihd** is only valid when **TA_MPRI(=0x02)** is set to **mbxatr**.

((**TA_TFIFO** || **TA_TPRI**) | (**TA_MFIFO** | **TA_MPRI**)) can be set in **mbxatr**. Mailbox waiting queue will be in the order of **FIFO** when **TA_TFIFO(=0x00)** is set, and it will be in the order of task priority when **TA_TPRI(=0x01)** is set. When **TA_MFIFO(=0x00)** is set, it will be in the order of **FIFO**, and when **TA_MPRI(=0x02)** is set, it will be in the order of message priority. When **NULL** is set to **mbrihd**, the configurator secures the necessary area.

4.7.2 cre_mbx create mailbox

Service Call

ER ercd=cre_mbx(ID mbxid,T_CMBX*pk_cmbx);

Parameter

ID	mbxid	ID number mailbox to create
T_CMBX*	pk_cmbx	Pointer to packet with mailbox creation information
Contents of pk_cmbx_Type: T_CMBX_		
ATR	mbxatr	mailbox attribute
PRI	maxmpri	maximum message priority
VP	mprihd	address for priority-based message queue header

Return Parameter

ER	ercd	Success E_OK or Error Code
-----------	-------------	-----------------------------------

Error Code

E_ID	invalid ID (mbxid is invalid or unusable)
E_NOMEM	lack of memory (cannot allocate message queue header)
E_RSATR	reserved attribute (mbxatr is invalid or unusable)
E_PAR	Parameter Error pk_cmbx,maxmpri,mprihd are invalid
E_OBJ	object state error (referent mailbox is already registered)
E_CTX	Called from Non-task Context

Function

Creates the mailbox that has the **ID** number set by **mbxid**, based on the mailbox creating information that has been set. **maxmpri** and **mprihd** is only valid when **TA_MPRI(=0x02)** is set to **mbxatr**. ((**TA_TFIFO** || **TA_TPRI**) | (**TA_MFIFO** | **TA_MPRI**)) can be set in **mbxatr**. Mailbox waiting queue will be in the order of **FIFO** when **TA_TFIFO(=0x00)** is set, and it will be in the order of task priority when **TA_TPRI(=0x01)** is set. Also, the mailbox of the message queue will be in the order for **FIFO** when **TA_MFIFO(=0x00)** is set, and when **TA_MPRI(=0x02)** is set, it will be in the order of message priority. When **NULL** is set to **mprihd**, the kernel secures the necessary area. Do not destroy the task creating information set by **pk_cmbx** when the task is in valid because the kernel uses this for task functional information.

4.7.3 acre_mbx create mailbox (ID number automatically assigned)_S_

Service Call

ER_ID mbxid=acre_mbx(T_CMBX*pk_cmbx);

Parameter

T_CMBX* **pk_cmbx** Pointer to packet with mailbox creation information
 Contents of **pk_cmbx_Type: T_CMBX_**
ATR **mbxatr** mailbox attribute
PRI **maxmpri** maximum message priority
VP **mprihd** address for priority-based message queue header

Return Parameter

ER_ID **mbxid** **ID** number (positive number) of created mailbox or error code

Error Code

E_NOID lack of **ID** number (no available mailbox **ID**)
E_NOMEM lack of memory (cannot allocate message queue header)
E_RSATR reserved attribute (**mbxatr** is invalid or unusable)
E_PAR Parameter Error **pk_cmbx,maxmpri,mprihd** are invalid_
E_CTX Called from Non-task Context

Function

Creates the mailbox based on the mailbox creating information set by **pk_cmbx**. When creating the **ID** number, the **ID** that has not been allocated in the mailbox management area is used. Please refer to **cre_mbx** for **pk_cmbx**.

4.7.4 **del_mbx** delete mailbox

Service Call

ER **ercd** = **del_mbx(ID mbxid);**

Parameter

ID **mbxid** **ID** number of delete object mailbox

Return Parameter

ER **ercd** Success_**E_OK**_or Error Code

Error Code

E_ID invalid **ID** (**mbxid** is invalid or unusable)

E_NOEX S object does not exist (referent mailbox is not registered)

E_CTX Called from Non-task Context

Function

Deletes the mailbox set by **mbxid**. When the kernel secures the message queue header accordingly to the priority, that area will be released.

4.7.5 **snd_mbx** send to mailbox **S_**

Service Call

ER ercd=snd_mbx(ID mbxid,T_MSG*pk_msg);

Parameter

ID **mbxid** ID number of mailbox to send
T_MSG* pk_msg address of message packet to be sent to mailbox

Return Parameter

ER **ercd** Success **E_OK** or Error Code

Error Code

E_ID invalid ID (**mbxid** is invalid or unusable)
E_NOEXS object does not exist (referent mailbox is not registered)
E_PAR Parameter Error **pk_msg** is invalid, the message priority inside the message packet (**msgpri** is invalid)
E_CTX Called from Non-task Context

Function

Sends the message with **pk_msg** as the starting address, to the mailbox set by **mbxid**. When there is tasks waiting for receive with the referent mailbox, it gives the starting address of the message packet set by **pk_msg**, and the task will be release from Wait. When there is no task waiting to receive, the message with **pk_msg** as the starting address is placed in the end of the message queue. Here, when **TA_MPRI(=0x02)** is set in the mailbox attribute, message is put in the message queue accordingly to its priority.

Message packet has to be defined by the user. How to illustrate in C language is as follows.

```
typedef struct
{
    T_MSG header;
    Mailbox itself
} T_MSG_PACKET;
```

Also, when **TA_MPRI** is set in mailbox attribute, the illustration will be as follows.

```
typedef struct
{
    T_MSG_PRI header;
    Mailbox itself
} T_MSG_PACKET_PRI;
```

Please call this service call using the starting address of the message packet created in these definitions, as **pk_msg**.

4.7.6 isnd_mbx send to mailbox (exclusive to Non-task context)

Service Call

ER ercd=isnd_mbx(**ID** mbxid,T_MSG***pk_msg**);

Parameter

ID **mbxid** **ID** number of mailbox to send
T_MSG* **pk_msg** address of message packet to be sent to mailbox

Return Parameter

ER **ercd** Success_**E_OK**_or Error Code

Error Code

E_ID invalid **ID** (**mbxid** is invalid or unusable)
E_NOEXS object does not exist (referent mailbox is not registered)
E_PAR Parameter Error_**pk_msg** is invalid, the message priority in the message packet
 (**msgpri** is invalid_
E_NOMEM Delayed Execution Buffer is full
E_CTX Called from task context

Function

Sends the message with **pk_msg** as the starting address, to the mailbox set by **mbxid**. When there is tasks waiting for receive with the referent mailbox, it gives the starting address of the message packet set by **pk_msg**, and the task will be release from Wait. When there is no task waiting to receive, the message with **pk_msg** as the starting address is placed in the end of the message queue. Here, when **TA_MPRI(=0x02)** is set in the mailbox attribute, message is put in the message queue accordingly to its priority.

4.7.7 rcv_mbx receive from mailbox_S_

Service Call

ER ercd=rcv_mbx(ID mbxid, T_MSG**ppk_msg);

Parameter

ID mbxid ID number of mailbox for receive

Return Parameter

ER ercd Success_E_OK_or Error Code
T_MSG* ppk_msg the head address of message packet from mailbox

Error Code

E_ID invalid ID (**mbxid** is invalid or unusable)
E_NOEXS object does not exist (referent mailbox is not registered)
E_PAR Parameter Error (**ppk_msg** is NULL)
E_RLWAI forced release from waiting state (**rel_wai** called while waiting state)
E_DLT object deleted while waiting (an object mailbox is deleted while task was in a waiting state)
E_CTX Called from Non-task Context

Function

Receives the message from mailbox set by **mbxid**, and stores the starting address in the area set by **ppk_msg**. When the message is in the message queue of the referent mailbox, then get the starting message packet. When there is no message, connect the invoking task to the Wait queue, and change its status to Receive Wait from mailbox.

4.7.8 prcv_mbx receive from mailbox (polling)_S_

Service Call

ER ercd=prcv_mbx(ID mbxid, T_MSG**ppk_msg);

Parameter

ID **mbxid** **ID** number of mailbox for receive

Return Parameter

ER **ercd** Success **E_OK** or Error Code
T_MSG* **pk_msg** the head address of message packet from mailbox

Error Code

E_ID invalid **ID** (**mbxid** is invalid or unusable)
E_NOEXS object does not exist (referent mailbox is not registered)
E_PAR parameter error (**ppk_msg** or **tmout** are invalid)
E_TMOUT polling failed or timeout
E_CTX Called from Non-task Context

Function

Receives the message from mailbox set by **mbxid**, and stores the starting address in the area set by **ppk_msg**. When the message is in the message queue of the referent mailbox, then get the starting message packet. When there is no message, **E_TMOUT** is returned.

4.7.9 trcv_mbx receive from mailbox (timeout)_S_

Service Call

ER ercd=trcv_mbx(**ID** mbxid,**T_MSG****ppk_msg,**TMO** tmout);

Parameter

ID	mbxid	ID number of mailbox for receive
TMO	tmout	specify timeout

Return Parameter

ER	ercd	Success_ E_OK or Error Code
T_MSG*	pk_msg	the head address of message packet from mailbox

Error Code

E_ID	invalid ID (mbxid is invalid or unusable)
E_NOEXS	object does not exist (referent mailbox is not registered)
E_PAR	parameter error (ppk_msg or tmout are invalid)
E_RLWAI	forced release from waiting state (rel_wai called while waiting state)
E_TMOUT	polling failed or timeout
E_DLT	object deleted while waiting (referent mailbox is deleted while task was in a waiting state)
E_CTX	Called from Non-task Context

Function

Receives the message from mailbox set by **mbxid**, and stores the starting address in the area set by **ppk_msg**. When the message is in the message queue of the referent mailbox, then get the starting message packet. When there is no message, connect the invoking task to the Wait queue, and change its status to Receive Wait from mailbox.

In **tmout**, **TMO_POL(=0)** and **TMO_FEVR(=-1)** can be set, addition to the positive value timeout time. When **TMO_POL** is set, it will act in the same way as **prcv_mbx**. Also, when **TMO_FEVR** is set, it will at in the same way as **rcv_mbx**.

4.7.10 **ref_mbx** reference state of mailbox

Service Call

ER **ercd** = **ref_mbx**(**ID mbxid**,**T_RMBX*pk_rmbx**);

Parameter

ID **mbxid** **ID** number of referent mailbox with information reference
T_RMBX* pk_rmbx pointer to a packet with mailbox state

Return Parameter

ER **ercd** Success **E_OK** or Error Code
Contents of **pk_rmbx** (Type: **T_RMBX**)
ID **wtskid** **ID** number of task at the head of the wait queue
T_MSG * pk_msg address of the message packet at head of message queue

Error Code

E_ID invalid **ID** (**mbxid** is invalid or unusable)
E_NOEXS object does not exist (referent mailbox is not registered)
E_PAR Parameter Error **NULL** has been set to **pk_rmbx*_**
E_CTX Called from Non-task Context

Function

Refers the information related to mailbox set by **mbxid**, and stores it in the area set by **pk_rmbx**. When there is no task in the referent mailbox Waiting queue, **TSK_NONE(=0)** will be set to **wtskid**.

4.8 Memory Pool Management Functions_Fixed-sized memory pools

4.8.1 CRE_MPF Create fixed-sized memory pool (static API)_S_

How to illustrate static API

```
CRE_MPF(ID mpfid,{ATR mpfatr,UINT blkcnt,UINT blksz,VP mpf});
```

Parameter

ID	mpfid	ID number of created fixed-sized memory pool
ATR	mpfatr	Fixed-sized memory pool attributes
UINT	blkcnt	Available memory blocks (number of UNITs)
UINT	blksz	Memory block size (in bytes)
VP	mpf	Beginning address of fixed-sized memory pool area

Function

Creates Fixed-sized memory pool that has the **ID** number set by **mpfid**, based on the Fixed-sized memory pool creating information that has been set. **mpfid** sets the automatic allocation non support integer value parameter, and **mpfatr** sets the preprocessor constant parameter.

(**TA_TFIFO** || **TA_TPRI**) can be set in **mpfatr**. Fixed-sized memory pool waiting queue will be in the order of **FIFO** when **TA_TFIFO(=0x00)** is set, and it will be in the order of task priority when **TA_TPRI(=0x01)** is set. When **NULL** is set to **mpf**, the configurator secures the necessary area.

4.8.2 cre_mpf Create fixed-sized memory pool

Service Call

ER ercd=cre_mpf(ID mpfid,T_CMPF*pk_cmpf);

Parameter

ID	mpfid	ID number of created fixed-sized memory pool
T_CMPF*	pk_cmpf	Pointer to fixed-sized memory pool information packet
Content of pk_cmpf_Type: T_CMPF_		
ATR	mpfatr	Fixed-sized memory pool attributes
UINT	blkcnt	Available memory blocks (number of UNITS)
UINT	blksz	Memory block size (in bytes)
VP	mpf	Beginning address of fixed-sized memory pool area

Return Parameter

ER **ercd** Success **E_OK** or Error Code

Error Code

E_ID	Error in ID number (mpfid is invalid or cannot be used)
E_NOMEM	Insufficient memory (unable to allocate memory pool area)
E_RSATR	Reserve attributes (error in mpfatr or unable to use)
E_PAR	Parameter error (pk_cmpf , blkcnt , blksz , mpf are invalid)
E_OBJ	Object Status error (fixed-sized memory pool is already registered)
E_CTX	Called from Non-task Context
E_SYS	System Error

Function

Creates Fixed-sized memory pool that has the **ID** number set by **mpfid**, based on the Fixed-sized memory pool creating information that has been set by **pk_cmpf**.
(**TA_TFIFO** || **TA_TPRI**) can be set in **mpfatr**. Fixed-sized memory pool waiting queue will be in the order of **FIFO** when **TA_TFIFO(=0x00)** is set, and it will be in the order of task priority when **TA_TPRI(=0x01)** is set. When **NULL** is set to **mpf**, the kernel secures the necessary area. Do not destroy the task creating information set by **pk_cmpf** when the task is in valid because the kernel uses this for task functional information.

4.8.3 acre_mpf Create fixed-sized memory pool (ID number automatically assigned)

Service Call

ER_ID mpfid=acre_mpf(T_CMPF*pk_cmpf);

Parameter

T_CMPF*	pk_cmpf	Pointer to fixed-sized memory pool information packet
Contents of pk_cmpf_Type: T_CMPF_		
ATR	mpfatr	Fixed-sized memory pool attributes
UINT	blkcnt	Available memory blocks (number of UNITS)
UINT	blksz	Memory block size (in bytes)
VP	mpf	Beginning address of fixed-sized memory pool area

Return Parameter

ER_ID	mpfid	ID number of fixed-sized memory pool (positive value) or error code
--------------	--------------	---

Error Code

E_NOID	Insufficient ID Numbers (Unable to assign to Fixed-sized Memory Pool ID)
E_NOMEM	Insufficient memory (unable to allocate memory pool area)
E_RSATR	Reserve attributes (error in mpfatr or unable to use)
E_PAR	Parameter error (pk_cmpf , blkcnt , blksz , mpf are invalid)
E_CTX	Called from Non-task Context
E_SYS	System Error

Function

Creates the Fixed-sized memory pool based on the Fixed-sized memory pool creating information set by **pk_cmpf**. When creating the **ID** number, the **ID** that has not been allocated in the Fixed-sized memory pool management area is used. Please refer to **cre_mpf** for **pk_cmpf**.

4.8.4 **del_mpf** Delete fixed-sized memory pool

Service Call

ER **ercd** = **del_mpf**(**ID mpfid**);

Parameter

ID **mpfid** **ID** number of fixed-sized memory pool to delete

Return Parameter

ER **ercd** Success_**E_OK**_or Error Code

Error Code

E_ID Error in **ID** number (**mpfid** is invalid or cannot be used)
E_NOEX S Object not created (target fixed-sized memory pool is not registered)
E_CTX Called from Non-task Context
E_SYS System Error

Function

Deletes the Fixed-sized memory pool set by **mpfid**. When the kernel secures the Fixed-sized memory pool area, then this area will be release.

4.8.5 **get_mpf** Get a fixed-sized memory block_S_

Service Call

ER **ercd**=**get_mpf**(**ID** **mpfid**,**VP*****p_blk**);

Parameter

ID **mpfid** **ID** number of target fixed-sized memory pool that allocates the memory block

Return Parameter

ER **ercd** Success **E_OK** or Error Code
VP **blk** Beginning address of allocated memory block

Error Code

E_ID Error in **ID** number (**mpfid** is invalid or cannot be used)
E_NOEXS Object not created (target fixed-sized memory pool is not registered)
E_PAR Parameter error (**p_blk**, is **NULL**)
E_RLWAI Compulsory release of wait status (accepted **rel_wai** during wait status)
E_DLT Deletion of wait object (delete target fixed-sized memory pool during wait status)
E_CTX Called from Non-task Context
E_OBJ Object Status Error
E_SYS System Error

Function

Get the memory block size set when creating Fixed-sized memory pool, from Fixed-sized memory pool set by **mpfid**, and store the starting address in the area set by **p_blk**. When there is no open memory block, place the invoking task in the Wait queue, and change the status to Fixed-sized memory block get wait.

4.8.6 pget_mpf Get a fixed-sized memory block. (polling)_S_

Service Call

ER **ercd**=**pget_mpf**(**ID** **mpfid**,**VP*****p_blk**);

Parameter

ID **mpfid** **ID** number of target fixed-sized memory pool that allocates the memory block

Return Parameter

ER **ercd** Success **E_OK** or Error Code
VP **blk** Beginning address of allocated memory block

Error Code

E_ID Error in **ID** number (**mpfid** is invalid or cannot be used)
E_NOEXS Object not created (target fixed-sized memory pool is not registered)
E_PAR Parameter error (**p_blk** is **NULL**)
E_TMOUT Polling failed
E_CTX Called from Non-task Context
E_OBJ Object Status Error
E_SYS System Error

Function

Get the memory block size set when creating Fixed-sized memory pool, from Fixed-sized memory pool set by **mpfid**, and store the starting address in the area set by **p_blk**. When there is no open memory block, **E_TMOUT** is returned.

4.8.7 tget_mpf Get a fixed-sized memory block. (timeout)_S_

Service Call

ER ercd=tget_mpf(ID mpfid,VP*p_blk,TMO tmout);

Parameter

ID	mpfid	ID of target fixed-sized memory pool that allocates the memory block
TMO	tmout	Speciry Timeout

Return Parameter

ER	ercd	Success E_OK or Error Code
VP	blk	Beginning address of allocated memory block

Error Code

E_ID	Error in ID number (mpfid is invalid or cannot be used)
E_NOEXS	Object not created (target fixed-sized memory pool is not registered)
E_PAR	Parameter error (p_blk , tmout are invalid)
E_RLWAI	Compulsory release of wait status (accepted rel_wai during wait status)
E_TMOUT	timeout
E_DLT	Deletion of wait object (delete target fixed-sized memory pool during wait status)
E_CTX	Called from Non-task Context
E_SYS	System Error

Function

Get the memory block size set when creating Fixed-sized memory pool, from Fixed-sized memory pool set by **mpfid**, and store the starting address in the area set by **p_blk**. When there is no open memory block, place the invoking task in the Wait queue, and change the status to Fixed-sized memory block get wait.

In **tmout**, **TMO_POL(=0)** and **TMO_FEVR(=-1)** can be set, addition to the positive value timeout time. When **TMO_POL** is set, it will act in the same way as **pget_mpf**. Also, when **TMO_FEVR** is set, it will at in the same way as **get_mpf**.

4.8.8 rel_mpf Release a fixed-sized memory block_S_

Service Call

ER ercd=rel_mpf(ID mpfid,VP blk);

Parameter

ID	mpfid	ID number for the fixed-sized memory pool where memory block is returned
VP	blk	Start address of released memory block

Return Parameter

ER	ercd	Success E_OK or Error Code
-----------	-------------	-----------------------------------

Error Code

E_ID	Invalid ID number (mpfid is invalid or cannot be used)
E_NOEXS	Object not created (target fixed-sized memory pool is not registered)
E_PAR starting	Parameter error (blk is invalid, return to different memory pool, return except the address of the allocated memory block)
E_CTX	Called from Non-task Context
E_SYS	System Error

Function

Releases the memory block with **blk** as the starting address, against the Fixed-sized memory pool set by **mpfid**.

When there is a task waiting to get the memory pool by the referent Fixed-sized memory pool, let the returned memory block get the first task in the Waiting queue, and release this task from Wait. The referent Fixed-sized memory pool that releases the memory block must be the same as the Fixed-sized memory pool that received the memory block. Also, the starting address of the memory block that is to be returned must be either **get_mpf**, **pget_mpf**, **tget_mpf** that are been returned as the starting address of the memory block that has been received, and has to be the one that is not been released. There is no assurance on how it operates, when the value other than stated above is set to **blk**.

4.8.9 ref_mpf Reference fixed-sized memory pool status

Service Call

ER **ercd** = ref_mpf(**ID mpfid**,**T_RMPF*pk_rmpf**);

Parameter

ID **mpfid** **ID** number for the fixed-sized return memory pool
T_RMPF* **pk_rmpf** Start address of released memory block

Return Parameter

ER **ercd** Success **E_OK** or Error Code
 Contents of **pk_rmpf** (Type: **T_RMPF**)
 ID **wtskid** Task **ID** number of task at head of wait queue
 UINT **fblkcnt** Number of available memory blocks (in no. of **UNITs**)

Error Code

E_ID Error in **ID** number (**mpfid** is invalid or cannot be used)
E_NOEXS Object not created (referent fixed-sized memory pool is not registered)
E_PAR Parameter Error **NULL** is set to **pk_rmpf*_**
E_CTX Called from Non-task Context
E_SYS System Error

Function

Refers the information related to Fixed-sized memory pool set by **mpfid**, and stores it in the area set by **pk_rmpf**. When there is no task in the referent Fixed-sized memory pool Waiting queue, **TSK_NONE(=0)** will be set to **wtskid**.

4.9 Memory Pool Management Functions _ Variable-sized Memory Pools

4.9.1 CRE_MPL Create variable-sized memory pool (Static API)

How to illustrate static API

```
CRE_MPL(ID mplid,{ATR mplatr,SIZE mplsz ,VP mpl});
```

Parameter

ID	mplid	ID number of target variable-sized memory pool
ATR	mplatr	Attributes of variable-sized memory pool
SIZE	mplsz	Variable-sized Memory block size (in bytes)
VP	mpl	Beginning address variable-sized memory pool area

Function

Creates Variable-sized memory pool that has the **ID** number set by **mplid**, based on the Variable-sized memory pool creating information that has been set. **mplid** sets the automatic allocation non support integer value parameter, and **mplatr** sets the preprocessor constant parameter.

(**TA_TFIFO** || **TA_TPRI**) can be set in **mplatr**. Variable-sized memory pool waiting queue will be in the order of **FIFO** when **TA_TFIFO(=0x00)** is set, and it will be in the order of task priority when **TA_TPRI(=0x01)** is set. When **NULL** is set to **mpl**, the configurator secures the necessary area.

4.9.2 cre_mpl Create variable-sized memory pool

Service Call

ER ercd=cre_mpl(ID mplid,T_CMPL*pk_cmpl);

Parameter

ID **mplid** ID number of target variable-sized memory pool to create
T_CMPL* pk_cmpl Pointer to packet containing variable-sized memory pool information.
 Contents of **pk_cmpl_Type: T_CMPL_**
ATR **mplatr** Attributes of variable-sized memory pool
SIZE **mplsz** Variable-sized Memory area size (in bytes)
VP **mpl** Beginning address variable-sized memory pool area

Return Parameter

ER **ercd** Success **E_OK** or Error Code

Error Code

E_ID Invalid **ID** number (**mplid** is invalid or unable to use)
E_NOMEM Insufficient memory (unable to allocate memory pool area)
E_RSATR Reserved attribute (**mplatr** is invalid, or unable to use)
E_PAR Parameter error (**pk_cmpl**, **mplsz**, **mpl** are invalid)
E_OBJ Object status error (target variable-sized memory pool is already registered)
E_CTX Called from Non-task Context
E_SYS System Error

Function

Creates Variable-sized memory pool that has the **ID** number set by **mplid**, based on the Variable-sized memory pool creating information that has been set by **pk_cmpl**.
(TA_TFIFO || TA_TPRI) can be set in **mplatr**. Variable-sized memory pool waiting queue will be in the order of **FIFO** when **TA_TFIFO(=0x00)** is set, and it will be in the order of task priority when **TA_TPRI(=0x01)** is set. When **NULL** is set to **mpl**, the kernel secures the necessary area. Do not destroy the task creating information set by **pk_cmpl** when the task is in valid because the kernel uses this for task functional information.

4.9.3 acre_mpl Create variable-sized memory pool (automatic ID number assigned)

Service Call

ER_ID **mplid**=**acre_mpl**(**T_CMPL*****pk_cmpl**);

Parameter

T_CMPL* **pk_cmpl** Pointer to packet containing variable-sized memory pool information
 Contents of **pk_cmpl_Type: T_CMPL_**
ATR **mplatr** Attributes of variable-sized memory pool
SIZE **mplsz** Variable-sized Memory area size (in bytes)
VP **mpl** Beginning address variable-sized memory pool area

Return Parameter

ER_ID **mplid** **ID** number for created variable-sized memory pool (positive value) or error code

Error Code

E_NOID Insufficient **ID** numbers (there are no variable-sized memory pool left to assign)
E_NOMEM Insufficient memory (unable to allocate memory pool area)
E_RSATR Reserved attribute (**mplatr** is invalid, or unable to use)
E_PAR Parameter error (**pk_cmpl**, **mplsz**, **mpl** are invalid)
E_CTX Called from Non-task Context
E_SYS System Error

Function

Creates the Variable-sized memory pool based on the Variable-sized memory pool creating information set by **pk_cmpl**. When creating the **ID** number, the **ID** that has not been allocated in the Variable-sized memory pool management area is used. Please refer to **cre_mpl** for **pk_cmpl**.

4.9.4 **del_mpl** Delete variable-sized memory pool

Service Call

ER **ercd** = **del_mpl**(**ID mplid**);

Parameter

ID **mplid** **ID** number of variable-sized memory pool to delete

Return Parameter

ER **ercd** Success **E_OK** or Error Code

Error Code

E_ID Invalid **ID** number (**mplid** is invalid or unable to use)
E_NOEXS Object is not created (target variable-sized memory pool is not registered)
E_CTX Called from Non-task Context
E_SYS System Error

Function

Deletes the Variable-sized memory pool set by **mplid**. When the kernel secures the Variable-sized memory pool area, then this area will be release.

4.9.5 **get_mpl** Allocate a variable-sized memory block

Service Call

ER ercd=get_mpl(**ID** mplid, **UINT** blksize, **VP***p_blk);

Parameter

ID	mplid	ID number of target variable-sized memory pool that allocates memory block
UINT	blksize	Allocated Memory block size (in byte)

Return Parameter

ER	ercd	Success E_OK or Error Code
VP	blk	Beginning address of allocated memory block

Error Code

E_ID	Invalid ID number (mplid is invalid or unable to use)
E_NOEXS	Object not created (target variable-sized memory pool is not registered)
E_PAR	Parameter error (NULL is set to p_blk)
E_RLWAI	Compulsory release of wait status (accepted rel_wai during wait status)
E_DLT	Deletion of pool (target Variable-sized memory pool was deleted during wait status)
E_CTX	Called from Non-task Context
E_OBJ	Object Status Error
E_SYS	System Error

Function

Get the memory block size set by **blksize**, from Variable-sized memory pool set by **mplid**, and store the starting address in the area set by **p_blk**. When there is no open memory block, place the invoking task in the Wait queue, and change the status to Variable-sized memory block get wait.

4.9.6 pget_mpl Allocate a variable-sized memory block (polling)

Service Call

ER ercd=pget_mpl(**ID** mplid, **UINT** blksize, **VP***p_blk);

Parameter

ID	mplid	ID number of target variable-sized memory pool that allocates memory block
UINT	blksize	Allocated Memory block size (in byte)

Return Parameter

ER	ercd	Success E_OK or Error Code
VP	blk	Beginning address of allocated memory block

Error Code

E_ID	Invalid ID number (mplid is invalid or unable to use)
E_NOEXS	Object not created (referent variable-sized memory pool is not registered)
E_PAR	Parameter error (NULL is set to p_blk)
E_TMOUT	Failed to poll or timeout
E_CTX	Called from Non-task Context
E_OBJ	Object Status Error
E_SYS	System Error

Function

Get the memory block size set by blksize, from Variable-sized memory pool set by **mplid**, and store the starting address in the area set by **p_blk**. When there is no open memory block, **E_TMOUT** is returned.

4.9.7 tget_mpl Allocate a variable-sized memory block (timeout)

Service Call

ER ercd=tget_mpl(**ID** mplid, **UINT** blksz, **VP***p_blk,**TMO** tmout);

Parameter

ID	mplid	ID number of target variable-sized memory pool that allocates memory block
UINT	blksz	Allocated Memory block size (in byte)
TMO	tmout	Specigy Timeout

Return Parameter

ER	ercd	Success E_OK or Error Code
VP	blk	Beginning address of allocated memory block

Error Code

E_ID	Invalid ID number (mplid is invalid or unable to use)
E_NOEXS	Object not created (referent variable-sized memory pool is not registered)
E_PAR	Parameter error (p_blk , tmout are invalid)
E_RLWAI	Compulsory release of wait status (accepted rel_wai during wait status)
E_TMOUT	Failed to poll or timeout
E_DLT	Deletion of pool (referent fixed-sized memory pool was deleted during wait status)
E_CTX	Called from Non-task Context
E_OBJ	Object Status Error
E_SYS	System Error

Function

Get the memory block size set by **blksz**, from Variable-sized memory pool set by **mplid**, and store the starting address in the area set by **p_blk**. When there is no open memory block, place the invoking task in the Wait queue, and change the status to Variable-sized memory block get wait.

In **tmout**, **TMO_POL(=0)** and **TMO_FEVR(=-1)** can be set, addition to the positive value timeout time. When **TMO_POL** is set, it will act in the same way as **pget_mpl**. Also, when **TMO_FEVR** is set, it will at in the same way as **get_mpl**.

4.9.8 rel_mpl Return variable-sized memory block

Service Call

ER ercd=rel_mpl(ID mplid,VP blk);

Parameter

ID	mplid	ID number of variable-sized memory pool that is returned
VP	blk	Beginning address of memory block to be returned

Return Parameter

ER	ercd	Success E_OK or Error Code
-----------	-------------	-----------------------------------

Error Code

E_ID	Invalid ID number (mplid is invalid or unable to use)
E_NOEXS	Unable to create object (Referent variable-sized memory pool is unregistered)
E_PAR starting	Parameter error (blk is invalid, return to different memory pool, return except the address of the allocated memory block)
E_CTX	Called from Non-task Context
E_SYS	System Error

Function

Releases the memory block with **blk** as the starting address, against the Variable-sized memory pool set by **mpfid**.

When there is a task waiting to get the memory pool by the referent Variable-sized memory pool, check whether the first task in the Wait queue that is trying to get the memory block size is enable or not, after releasing the memory block. When the task is enabling to get the memory block size, then release this task from Wait. The referent Variable-sized memory pool that releases the memory block must be the same as the Variable-sized memory pool that received the memory block. Also, the starting address of the memory block that is to be returned must be either **get_mpl**, **pget_mpl**, **tget_mpl** that are been returned as the starting address of the memory block that has been received, and has to be the one that is not been released. There is no assurance on how it operates, when the value other than stated above is set to **blk**.

4.9.9 **ref_mpl** **Reference variable-sized memory pool status**

Service Call

ER **ercd** = **ref_mpl**(**ID** **mplid**,**T_RMPL*****pk_rmpl**);

Parameter

ID **mplid** **ID** number of target variable-sized memory pool
T_RMPL* **pk_rmpl** Pointer to the packet that returns the variable-sized memory pool status

Return Parameter

ER **ercd** Success **E_OK** or Error Code
 Contents of **pk_rmpl** (Type: **T_RMPL**)
 ID **wtskid** **ID** number of task at head of wait queue
 SIZE **fmplsz** Total size of open area of variable-sized memory pool (in bytes)
 UINT **fblksz** Maximum available memory block (in bytes)

Error Code

E_ID Invalid **ID** number (**mplid** is invalid or unable to use)
E_NOEXS Unable to create object (referent variable-sized memory pool is unregistered)
E_PAR Parameter error **_NULL** is set to **pk_rmpl*_**
E_CTX Called from Non-task Context
E_SYS System Error

Function

Refers the information related to Variable-sized memory pool set by **mplid**, and stores it in the area set by **pk_rmpl**. When there is no task in the referent Variable-sized memory pool Waiting queue, **TSK_NONE(=0)** will be set to **wtskid**.

4.10 System Time Management

4.10.1 **set_tim** Set system time_S_

Service Call

ER ercd=set_tim(SYSTEM*p_system);

Parameter

SYSTEM system Time to set

Return Parameter

ER ercd Success_**E_OK**_or Error Code

Error Code

E_PAR Parameter error_**p_system** is invalid_
E_CTX Called from Non-task Context

Function

Sets the current system time to the time stored in the area set be **p_system**.

4.10.2 **get_tim** Reference system time **_S_**

Service Call

ER ercd=get_tim(SYSTIM*p_system);

Parameter

None

Return Parameter

ER	ercd	Success _E_OK_ or Error Code
SYSTIM	system	Present system time

Error Code

E_PAR	Parameter error _p_system is invalid
E_CTX	Called from Non-task Context

Function

Calls the current system time and stores it in the area set by **p_system**.

4.10.3 vget_tim Reference system time

Service Call

SYSTIM system=vget_tim();

Parameter

None

Return Parameter

SYSTIM **system** Current system time

Error Code

None

Function

Returns the current System time. This service call can be called from both Task Context and Non Task Context. This service call is original specification.

4.10.4 CRE_CYC Create Cyclic handler (Static API)_S_

How to illustrate static API

```
CRE_CYC(ID cycid,{ATR cycatr,VP_INT exinf,FPcychdr,RELTIM cyctim,RELTIM cycphs});
```

Parameter

ID	cycid	ID number of the referent Cyclic handler to be created
ATR	cycatr	Cyclic Handler Attribute
VP_INT	exinf	Extended information of the cyclic handler
FP	cychdr	Activation address of the cyclic handler
RELTIM	cyctim	Activation cycle of the cyclic handler
RELTIM	cycphs	Activation phase of the cyclic handler

Function

Creates Cyclic handler that has the **ID** number specified by **cycid**, based on the Cyclic handler creating information that has been set. **cycid** sets the automatic allocation non support integer value parameter, and **cycatr** sets the preprocessor constant parameter. ((**TA_HLNG** | [**TA_STA**] | [**TA_PHS**])) can be set in **cycatr**.

Cyclic handler is activated with the High Language Interface in setting **TA_HLNG(0x00)**. Since this version does not support assembler language interface in executing the Task Exception Handling routine, this setting could be omitted.

When **TA_STA(0x02)** is set, Cyclic handler will be activated after it is created. When it is not set, Cyclic handler will not be activated. When **TA_PHS(=0x04)** is set, it determines the time that will be activated the next by keeping the activating phase of the Cyclic handler, before the Cyclic handler is activated. The time that first activates the Cyclic handler should be the time where activating phase that has been set, is added in the system initialized time.

4.10.5 cre_cyc Create Cyclic handler

Service Call

ER **ercd**=**cre_cyc**(**ID** **cycid**,**T_CCYC*****pk_ccyc**);

Parameter

ID	cycid	ID number of the Cyclic Handler to be created
T_CCYC*	pk_ccyc	Pointer to packet containing creation data
Contents of pk_ccyc Type: T_CCYC_		
ATR	cycatr	Cyclic Handler Attribute
VP_INT	exinf	Extended information of the cyclic handler
FP	cychdr	Activation address of the cyclic handler
RELTIM	cycetim	Activation cycle of the cyclic handler
RELTIM	cycphs	Activation phase of the cyclic handler

Return Parameter

ER **ercd** Success **E_OK** or Error Code

Error Code

E_ID	Invalid ID number (invalid cycid or unable to use)
E_RSATR	Reserved attribute (invalid cycatr or not able to use)
E_PAR	Parameter error (pk_ccyc , cychdr , cycetim , cycphs are invalid)
E_OBJ	Object status error (Target cyclic handler is already registered)
E_CTX	Called from Non-task Context
E_SYS	System Error

Function

Creates Cyclic handler that has the **ID** number set by **cycid**, based on the Cyclic handler creating information that has been set. **cycid** sets the automatic allocation non support integer value parameter, and **cycatr** sets the preprocessor constant parameter. ((**TA_HLNG** | [**TA_STA**] | [**TA_PHS**]) can be set in **cycatr**.

Cyclic handler is activated with the High Language Interface in setting **TA_HLNG(0x00)**. Since this version does not support assembler language interface in executing the Task Exception Handling routine, this setting could be omitted.

When **TA_STA(0x02)** is set, Cyclic handler will be activated after it is created. When it is not set, Cyclic handler will not be activated. When **TA_PHS(=0x04)** is set, it determines the time that will be activated the next by keeping the activating phase of the Cyclic handler, before the Cyclic handler is activated. The time that first activates the Cyclic handler should be the time where activating phase that has been set, is added in this service call. When a value bigger than **cycetim** is set to **cycphs**, **E_PAR** is returned. Do not destroy the task creating information set by **pk_ccyc** when the task is in valid because the kernel uses this for task functional information.

4.10.6 acre_cyc Create Cyclic handler (ID number automatically assigned)

Service Call

ER_ID cycid=acre_cyc(T_CCYC*pk_ccyc);

Parameter

T_CCYC*	pk_ccyc	Pointer to the packet containing creating data
Contents of pk_ccyc_Type: T_CCYC_		
ATR	cycatr	Cyclic Handler Attribute
VP_INT	exinf	Extended information of the cyclic handler
FP	cychdr	Activation address of the cyclic handler
RELTIM	cyctim	Activation cycle of the cyclic handler
RELTIM	cycphs	Activation phase of the cyclic handler

Return Parameter

ER_ID	cycid	ID number of created cyclic handler (positive number) or error code
--------------	--------------	---

Error Code

E_NOID	ID number shortage (cyclic handler ID that could be allocated is not available)
E_RSATR	Reserved attribute (invalid cycatr or not able to use)
E_PAR	Parameter error (pk_ccyc , cychdr , cyctim , cycphs are invalid)
E_CTX	Called from Non-task Context
E_SYS	System Error

Function

Creates the Cyclic handler that has the **ID** number set by **cycid**, based on the Cyclic Handler creating information set by **pk_ccyc**. When creating the **ID** number, the **ID** that has not been allocated in the Cyclic handler management area is used. Please refer to **cre_cyc** for **pk_ccyc**.

4.10.7 **del_cyc** Delete cyclic handler

Service Call

ER **ercd** = **del_cyc**(**ID** **cycid**);

Parameter

ID **cycid** **ID** number of cyclic handler to delete

Return Parameter

ER **ercd** Success_**E_OK**_or Error Code

Error Code

E_ID	Invalid ID number (cycid is invalid or unable to use)
E_NOEX S	Object not yet created (target cyclic handler is not registered)
E_CTX	Called from Non-task Context
E_SYS	System Error

Function

Deletes the Cyclic handler set by **cycid**.

4.10.8 **sta_cyc** Start cyclic handler operation_S_

Service Call

ER **ercd**=**sta_cyc**(**ID** **cycid**);

Parameter

ID **cycid** **ID** number of the referent cyclic handler to be started

Return Parameter

ER **ercd** Success_**E_OK**_or Error Code

Error Code

E_ID Invalid **ID** number (**cycid** is invalid or unable to use)
E_NOEXS Object not yet created (target cyclic handler is not registered)
E_CTX Called from Non-task Context
E_SYS System Error

Function

Changes the Cyclic handler set by **cycid** to activating status.

When **TA_PHS** is not set in the Cyclic handler attribute, the time that will activate the next Cyclic handler will be the time where Cyclic handler activating cycle is added in the time this service call is called.

When the Cyclic handler that is activating without **TA_PHS** has been set to the Cyclic handler attribute in the referent Cyclic handler, it only resets the time that will be activated next by the Cyclic handler. Nothing will be done when the Cyclic handler is activating with the setting by **TA_PHS** in the Cyclic handler attribute.

4.10.9 **stp_cyc** Stop Cyclic handler operation_S_

Service Call

ER **ercd**=stp_cyc(**ID** **cycid**);

Parameter

ID **cycid** **ID** number of target cyclic handler to stop

Return Parameter

ER **ercd** Success_**E_OK**_or Error Code

Error Code

E_ID Invalid **ID** number (**cycid** is invalid or unable to use)
E_NOEXS Object not yet created (target cyclic handler is not registered)
E_CTX Called from Non-task Context
E_SYS System Error

Function

Changes the Cyclic handler set by **cycid** to stop.

4.10.10 ref_cyc Reference the status of a cyclic handler

Service Call

ER **ercd** = ref_cyc(**ID** cycid, **T_RCYC*****pk_rcyc**);

Parameter

ID **cycid** **ID** number of cyclic handler to reference
T_RCYC* **pk_rcyc** Pointer to the packet that returns cyclic handler status

Return Parameter

ER **ercd** Success **E_OK** or Error Code
 Contents of **pk_rcyc** (Type: **T_RCYC**)
 STAT **cycstat** Cyclic handler operational state
 RELTIM **lefttim** Time left before the next activation of cyclic handler

Error Code

E_ID Invalid **ID** number (**cycid** is invalid or unable to use)
E_NOEXS Object not yet created (target cyclic handler is non-registered)
E_PAR Parameter error **NULL** is set to **pk_rcyc***_
E_CTX Called from Non-task Context

Function

Refers the information related to Cyclic handler set by **cycid**, and stores it in the area set by **pk_rcyc**.
lefttim will be **0** when the referent Cyclic handler is not activating.

4.11 Time management function Alarm Handler

4.11.1 CRE_ALM Create Alarm Handler_Static API_

How to illustrate Static API

```
CRE_ALM(ID almid, {ATR almatr, VP_INT exinf, FP almhdr});
```

Parameter

ID	almid	ID number of the referent alarm handler to be created
ATR	almatr	alarm handler attribute
VP_INT	exing	extended information of the alarm handler
FP	almhdr	activating address of the alarm handler

Function

Create the alarm handler that has the **ID** number specified by **almid**, based on the alarm handler creating information that has been specified. **almid** sets the automatic allocation non support integer value parameter, and **almatr** sets the preprocessor constant parameter.

TA_HLNG can be set to **almatr**. The setting of **TA_HLNG(0x00)** executes the alarm handler using High Level Language interface. Since executing the alarm handler using the assembler language interface is not supported in this version, you can omit this setting.

4.11.2 cre_alm Create Alarm Handler

Service Call

ER **ercd**=**cre_alm**(**ID** **almid**, **T_CALM*** **pk_calm**);

Parameter

ID	almid	ID number of the referent alarm handler to be created
T_CALM*	pk_calm	pointer to the packet with alarm handler creating information
Contents of Pk_calm (T_CALM type)		
ATR	almatr	alarm handler attribute
VP_INT	exinf	extended information of the alarm handler
FP	almhdr	activating address of the alarm handler

Return Parameter

ER **ercd** Success_ **E_OK**_or Error Code

Error Code

E_ID	Invalid ID number (almid is invalid or cannot be used)
E_RSATR	Reserved Attribute (almatr is invalid or cannot be used)
E_PAR	Parameter Error (pk_calm , almhdr are invalid)
E_OBJ	Object status error (referent alarm handler has already been registered)
E_CTX	Called from the Non-task Context
E_SYS	System Error

Function

Create the alarm handler that has the **ID** number specified by **almid**, based on the alarm handler creating information that has been specified by **pk_calm**. **almid** sets the automatic allocation non support integer value parameter, and **almatr** sets the preprocessor constant parameter.

TA_HLNG can be set to **almatr**. The setting of **TA_HLNG(0x00)** executes the alarm handler using High Level Language interface. Since executing the alarm handler using the assembler language interface is not supported in this version, you can omit this setting. The kernel uses the alarm handler creating information specified by **pk_calm** for alarm handler behavior information, please do not delete appropriate alarm handler when it is still valid.

4.11.3 acre_alm create alarm handler (ID number automatically allocated)

Service Call

ER_ID almid=acre_alm(**T_CALM*pk_calm**);

Parameter

T_CALM*pk_calm Pointer to the packet with alarm handler creating information
 Contents of **pk_calm** (**T_CALM Type**)
 ATR almatr alarm handler attribute
 VP_INT exinf extended information of the alarm handler
 FP almhdr activating address of the alarm handler

Return Parameter

ER_ID almid **ID** number (positive value) of the created alarm handler of the error code

Error Code

E_NOID Not enough **ID** number (No alarm handler **ID** in the possibly allocated)
E_RSATR Reserved Attribute (**almatr** is invalid or cannot be used)
E_PAR Parameter Error (**pk_calm**, **almhdr** are invalid)
E_CTX Called from the Non-task Context
E_SYS System Error

Function

Create the alarm handler that has the **ID** number specified by **almid**, based on the alarm handler creating information that has been specified by **pk_calm**. The **ID** number that is created will allocate the **ID** that has not yet allocated in the alarm handler management area.
 Please refer to **cre_alm** for the details on **pk_calm**.

4.11.4 del_aim Delete alarm handler

Service Call

ER **ercd=del_alm(ID almid);**

Parameter

ID **almid** **ID** number of the referent alarm handler to be deleted

Return Parameter

ER **ercd** Success_**E_OK**_or Error Code

Error Code

E_ID Invalid **ID** number (**almid** is invalid or cannot be used)
E_NOEXS Object not created (referent alarm handler is not registered)
E_CTX Called from the Non-task Context
E_SYS System Error

Function

Deletes the alarm handler specified by **almid**.

4.11.5 sta_alm Start the alarm handler

Service Call

ER ercd=sta_alm(**ID** almid, **RELTIM** almtim);

Parameter

ID almid **ID** number of the referent alarm handler that is to be started
RELTIM almtim activating time of the alarm handler (relative time)

Return Parameter

ER ercd Success (**E_OK**) or Error Code

Error Code

E_ID Invalid **ID** number (**cyaid** is invalid or cannot be used)
E_NOEXS Object not created (referent alarm handler is not created)
E_PAR Parameter Error (**almtim** are invalid)
E_CTX Called from the Non-task Context
E_SYS System Error

Function

Start activating the alarm handler by setting the alarm handler activating time specified by **almid**, which needs to be set from the time the service call has been called to the point after the relative time specified by **almtim**. When the the alarm handler that has already been activated is set to the referent alarm handler, release the previous activating time and set te new activating time.

4.11.6 ista_alm Start the alarm handler (exclusive to Non-task Context)

Service Call

ER ercd=ista_alm(**ID** almid, **RELTIM** almtim);

Parameter

ID almid **ID** number of the referent alarm handler to be activated
RELTIM almtim Activating time of the alarm handler (relative time)

Return Parameter

ER ercd Success (**E_OK**) or Error Code

Error Code

E_ID Invalid **ID** number (**cyid** is invalid or cannot be used)
E_NOEXS Object not created (referent alarm handler is not created)
E_PAR Parameter Error (**almtim** are invalid)
E_CTX Called from the Non-task Context
E_NOMEM Delayed Executin Buffer is full

Function

Start activating the alarm handler by setting the alarm handler activating time specified by **almid**, which needs to be set from the time the service call has been called to the point after the relative time specified by **almtim**. When the the alarm handler that has already been activated is set to the referent alarm handler, release the previous activating time and set te new activating time.

This service call is a unique specification. Also, the service call for Non-task Context will be executed with delay, so the specified acitivating time will be from when the delayed execution starts until the relative time.

4.11.7 **stp_alm** **Stop the alarm handler**

Service Call

ER **ercd=stp_alm(ID almid)**

Parameter

ID **almid** **ID** number of the referent alarm handler to be stopped

Return Parameter

ER **ercd** Success (**E_OK**) or Error Code

Error Code

E_ID	Invalid ID number (almid is invalid or cannot be used)
E_NOEXS	Object not created (referent alarm handler is not created)
E_CTX	Called from the Non-task Context
E_SYS	System Error

Function

Release the activating time of the alarm handler specified by **almid** and stops the alarm handler.

4.11.8 **istp_alm** Stops the alarm handler (exclusive to Non-task Context)

Service Call

ER **ercd**=**istp_alm**(**ID** **almid**)

Parameter

ID **almid** **ID** number of the referent alarm handler to be stopped

Return Parameter

ER **ercd** Success (**E_OK**) or Error Code

Error Code

E_ID	Invalid ID number (almid is invalid or cannot be used)
E_NOEXS	Object not created (referent alarm handler is not created)
E_CTX	Called from the Non-task Context
E_NOMEM	Delayed Execution Buffer is full

Function

Release the activating time of the alarm handler specified by **almid** and stops the alarm handler.

This service call is a unique specification. Also, the service call for Non-task Context will be executed with delay, so the specified activating time will be from when the delayed execution starts until the relative time.

4.11.9 ref_alm Alarm handler status reference

Service Call

ER **ercd**=ref_alm(**ID** **almid**, **T_RALM*****pk_ralm**);

Parameter

ID	almid	ID number of the referent alarm handler status reference
T_RALM*	pk_ralm	Pointer to the packet that returns the alarm handler status

Return Parameter

ER	ercd	Success (E_OK) or Error Code
Contents of pk_ralm (T_RALM Type)		
STAT	almstat	Activating status of the alarm handler
RELTIM	lefttim	Time until the alarm handler activating time

Error Code

E_ID	Invalid ID number (cycid is invalid or cannot be used)
E_NOEXS	Object not created (referent alarm handler is not created)
E_CTX	Called from the Non-task Context
E_PAR	Parameter Error (NULL is set to pk_rcyc*)
E_SYS	System Error

Function

Refers the status on the alarm handler specified by **almid** and stores it in the area specified by **pk_ralm**.
When the referent alarm handler is not activating, the **lefttim** will be **0**.

4.12 System state management function

4.12.1 **rot_rdq** Rotation of priority of task_S_

Service Call

ER ercd=rot_rdq(PRI tskpri);

Parameter

PRI tskpri Priority of object at priority of rotation

Return Parameter

ER ercd Success **E_OK** or Error Code

Error Code

E_PAR Parameter error **tskpri** is invalid_
E_CTX Called from Non-task Context

Function

Rotates the priority of the task priority set by **tskpri**.
When **TPRI_SELF(=0)** is set to **tskpri**, the base priority of the invoking task will be the referent priority.

4.12.2 irot_rdq Rotation of priority of task (exclusive to Non-task context)_S_

Service Call

ER **ercd**=irot_rdq(**PRI** **tskpri**);

Parameter

PRI **tskpri** Priority of object at priority of rotation

Return Parameter

ER **ercd** Success_**E_OK**_or Error Code

Error Code

E_PAR Parameter error_**tskpri** is invalid_
E_CTX Called from task context
E_NOMEM Delayed Execution Buffer Full

Function

Rotates the priority of the task priority set by **tskpri**.
TPRI_SELF(=0) cannot be set in **tskpri**.

4.12.3 **get_tid** Reference of task ID under execution (exclusive to Non-task context)_S_

Service Call

ER **ercd**=**get_tid**(**ID*****p_tskid**);

Parameter

None

Return Parameter

ER **ercd** Success_**E_OK**_or Error Code

ID* **p_tskid** Pointer to the parameter that stores the referent Task **ID** number under execution.
If any of the task prepared by the application cannot be executed, **TSK_NONE** is returned.

Error Code

E_CTX Called from Non-task Context

E_PAR Parameter Error (**p_tskid** is invalid)

Function

Refers the **ID** number of the invoking task, and stores it in the area set by **p_tskid**.

4.12.4 iget_tid task ID Reference under execution_S_

Service Call

ER ercd=iget_tid(ID*p_tskid);

Parameter

None

Return Parameter

ER **ercd** Success_ **E_OK** or Error Code

ID* **p_tskid** Pointer to the parameter that stores the referent Task **ID** number under execution.
If any of the task prepared by the application cannot be executed, **TSK_NONE** is returned.

Error Code

E_CTX Called from task context

E_PAR Parameter Error (**p_tskid** is invalid)

Function

Refers the **ID** number of the activating task, and stores it in the area set by **p_tskid**.

4.12.5 vget_tid Reference of the activating task ID (exclusive to Task Context)

Service Call

ER_ID ercd=vget_tid();

Parameter

None

Return Parameter

ER_ID **ercd** Success_Invoking task **ID**_or Error Code

Error Code

E_CTX Called from Non-task context

E_NOID Could not detect the task **ID**

Function

Returns the **ID** number of the invoking task.
This service call is an original specification.

4.12.6 **loc_cpu** Shift to CPU lock state_S_

Service Call

ER ercd=loc_cpu();

Parameter

None

Return Parameter

ER ercd Success_E_OK_or Error Code

Error Code

E_CTX Called from Non-task Context

Function

Change to **CPU** Lock status. Nothing will be done when it is called as **CPU** lock status.

4.12.7 **iloc_cpu** Shift to CPU lock state (exclusive to Non-task context)_S_

Service Call

ER **ercd=iloc_cpu();**

Parameter

None

Return Parameter

ER **ercd** Success_**E_OK**_or Error Code

Error Code

E_CTX Called from task context

Function

Change to **CPU** Lock status. Nothing will be done when it is called as **CPU** lock status.

4.12.8 **unl_cpu** Cancellation of CPU lock state_S_

Service Call

ER ercd=unl_cpu();

Parameter

None

Return Parameter

ER ercd Success_**E_OK****_or Error Code**

Error Code

E_CTX Called from Non-task Context

Function

Change to **CPU** Lock Release status. Nothing will be done when it is called as **CPU** lock Release status.

4.12.9 iunl_cpu Cancellation of CPU lock state (exclusive to Non-task context)_S_

Service Call

ER ercd=iunl_cpu();

Parameter

None

Return Parameter

ER **ercd** Success_**E_OK**_or Error Code

Error Code

E_CTX Called from task context

Function

Change to **CPU** Lock Release status. Nothing will be done when it is called as **CPU** lock Release status.

4.12.10 **dis_dsp** Disable dispatch_S_

Service Call

ER ercd=dis_dsp();

Parameter

None

Return Parameter

ER ercd Success_**E_OK****_or Error Code**

Error Code

E_CTX Called from Non-task Context

Function

Changes to Dispatch disable status. Nothing will be done when it is called as Disable dispatch.

4.12.11 ena_dsp Enable dispatch_S_

Service Call

ER ercd=ena_dsp();

Parameter

None

Return Parameter

ER **ercd** Success_**E_OK**_or Error Code

Error Code

E_CTX Called from Non-task Context

Function

Changes to Dispatch enable status. Nothing will be done when it is called as Disable enable.

4.12.12 **sns_ctx** Sense context_S_

Service Call

BOOL state=sns_ctx();

Parameter

None

Return Parameter

BOOL state context

Function

Return **TRUE** when called by Non Task Context, and return **FALSE** when called from Task Context.

4.12.13 sns_loc Reference CPU locked state_S_

Service Call

BOOL state=sns_loc();

Parameter

None

Return Parameter

BOOL state CPU locked state

Function

Return **TRUE** when the system is in CPU Lock Status, and return **FALSE** when the system is in CPU Lock Release status.

4.12.14 sns_dsp Reference dispatch disabled state_S_

Service Call

BOOL state=sns_dsp();

Parameter

None

Return Parameter

BOOL **state** dispatch disabled state

Function

Return **TRUE** when the system is in Dispatch Disable Status, and return **FALSE** when the system is in Dispatch Enable status.

4.12.15 sns_dpn Reference dispatch pending state_S_

Service Call

BOOL state=sns_dpn();

Parameter

None

Return Parameter

BOOL **state** dispatch pending state

Function

Return **TRUE** when the system is in Dispatch pending Status, and return **FALSE** when it is not.

4.12.16 DEF_INH Define interrupt handler (static API)_S_

How to illustrate static API

```
DEF_INH=(INHNO inhno,{ATR inhatr,FP inthdr});
```

Parameter

INHNO	inhno	Referent Interrupt handler number to be defined
ATR	inhatr	Interrupt handler attribute
FP	inthdr	Start address of interrupt handler

Function

Defines the Interrupt handler on the Interrupt handler number set by **inhno**, based on the Interrupt handler define information that has been set. **inhno** sets the automatic allocation non support integer value parameter, and **inhatr** sets the preprocessor constant parameter. Please refer to the target specific manual for **inhno** is CPU and target dependent.

4.12.17 **def_inh** Define interrupt handler

Service Call

ER **ercd**=def_inh(INHNO inhno,T_DINH*pk_dinh);

Parameter

INHNO	inhno	Interrupt handler number to define
T_DINH*	pk_dinh	pointer to the packet with interrupt handler definition information
Contents of pk_dinh_Type: T_DINH_		
ATR	inhatr	Interrupt handler attribute
FP	inthdr	Start address of interrupt handler

Return Parameter

ER **ercd** Success **E_OK** or Error Code

Error Code

E_RSATR	Reserved attribute (inhatr is invalid or appropriate)
E_PAR	Parameter error inhno, pk_dinh, inthdr are invalid_
E_CTX	Called from Non-task Context
E_SYS	System Error

Function

Defines the Interrupt handler on the Interrupt handler number set by **inhno**, based on the Interrupt handler define information set by **pk_dinh**. Please refer to the target specific manual for **inhno** is **CPU** and target dependent.

4.12.18 DEF_EXC Define CPU exception handler (static API)_S_

How to illustrate static API

```
DEF_EXC(EXCNO excno,{ATR excatr,FP exchdr});
```

Parameter

EXCNO	excno	Number of CPU exception handler to be defined
ATR	excatr	CPU exception handler attribute
FP	exchdr	Start address of CPU exception handler

Function

Defines the CPU Exception handler on the CPU Exception handler number set by **excno**, based on the CPU Exception handler define information that has been set. **excno** sets the automatic allocation non support integer value parameter, and **inhatr** sets the preprocessor constant parameter. Please refer to the target specific manual for **excno** is **CPU** and target dependent.

4.12.19 **def_exc** Define CPU exception handler

Service Call

ER **ercd**=**def_exc**(**EXCNO** **excno**,**T_DEXC*****pk_dexc**);

Parameter

EXCNO **excno** Number of **CPU** exception handler to be defined
T_DEXC* **pk_dexc** Pointer to the packet with exception handler definition
 Contents of **pk_dexc** Type: **T_DEXC**
ATR **excatr** **CPU** exception handler attribute
FP **exchdr** Start address of **CPU** exception handler

Return Parameter

ER **ercd** Success **E_OK** or Error Code

Error Code

E_RSATR Reserved attribute (**excno**, **pk_dexc**, **exchdr** are invalid)
E_PAR Parameter error **excno**,**pk_dexc**,**exchdr** are invalid_
E_CTX Called from Non-task Context
E_SYS System Error

Function

Defines the CPU Exception handler on the CPU Exception handler number set by **excno**, based on the CPU Exception handler define information that has been set. Please refer to the target specific manual for **excno** is **CPU** and target dependent.

4.12.20 ref_cfg Reference configuration information

Service Call

```
ER ercd=ref_cfg(T_RCFG *pk_rcfg);
```

Parameter

T_RCFG* **pk_rcfg** Pointer to the packet with configuration information

Return Parameter

ER **ercd** Success_ **E_OK** _or Error Code

T_RCFG* **pk_rcfg** Pointer to the packet with configuration information

Contents of **pk_rcfg_Type: T_RCFG_**

INT tsk_max	maximum number of tasks
INT sem_max	maximum number of semaphore
INT flg_max	maximum number of eventflags
INT dtq_max	maximum number of data queue
INT mbx_max	maximum number of mailbox
INT mtx_max	maximum number of mutex (always 0 in the current version)
INT mbf_max	maximum number of message buffer (always 0 in the current version)
INT por_max	maximum number of rendezvous port (always 0 in the current version)
INT mpf_max	maximum number of fixed-size memory pool
INT mpl_max	maximum number of variable-sized memory pool
INT cyc_max	maximum number of cyclic handler
INT alm_max	maximum number of alarm handler (always 0 in the current version)
UW cfg_dat	configuration date
UW cfg_tim	configuration time
SIZE kernel_memory_size	kernel usage memory pool size

Error Code

E_PAR Parameter error_ **pk_rcfg** is invalid_

Function

Refers the information set by configuration or static information of the system, and stores it in the area set by **pk_rcfg**.

4.12.21 ATT_INI Attach initialization routine (static API_S_

How to illustrate static API

```
ATT_INI({ATR iniatr,VP_INT,FP inirtn});
```

Parameter

ATR	iniatr	Attribute of initialization routine
VP_INT	exinf	extended information of initialization routine
FP	inirtn	start address of initialization routine

Function

Adds the Initialize routine based on each parameter that has been set. **iniatr** is preprocessor constant parameter. The initialize routine that has been added will activate as a part of static **API** processing while the system is initialized.

5 Error during Runtime

5.1 System Error

When the fatal error occurs, where the **OS** is not been able to continuously activate, it goes into an infinite loop within a function called **ERC_System_Error**, after writing the Error Code in the extensive variable called **ERD_Error_Code**.

When this loop is detected, you may check the cause by looking into **ERD_Error_Code**.

Value of **ERD_Error_Code**

1: Failed in creating the system timer handler (error during system activate)

4: Interrupt that cannot be handled is generated

100: Error occurred during the static **API** process

Also when each service call returns **E_SYS** error, the error occurring status will be recorded in the following extensive parameter, so you can refer to these parameters to check the cause.

ID_kernel_ESEYS_objid;	Object ID that you tried to refer
ID_kernel_ESYS_tskid;	Task ID that the error occurred
UNSIGNED_kernel_ESYS_stat;	Internal Error code
Char_kernel_ESYS_file[1024];	Kernel file name that the error occurred
Int_kernel_ESYS_line;	Line number of the kernel file

6 Configurator

Tested Environment Windows95/98/2000/NT4

Configurator _ mip_Cfg.exe
Configurator for Windows _ mip_CfgW.exe

Executing **mip_Cfg.exe** processes **mip_CfgW**. Because of this, **mip_CfgW.exe** and **mip_Cfg.exe** must be copied in the same folder or in the folder that is specified by the Path environmental variable. (We recommend copying it in the same folder.) Furthermore, the actual work folder will be the folder that has the configuration file specified by the **Setting** dialog. So, unless you do not specify the file name of an output file including the folder name, it will be output in the same folder as the configuration file. Please refer to the command line argument to **mip_Cfg.exe** for the details of the **Setting** dialog.

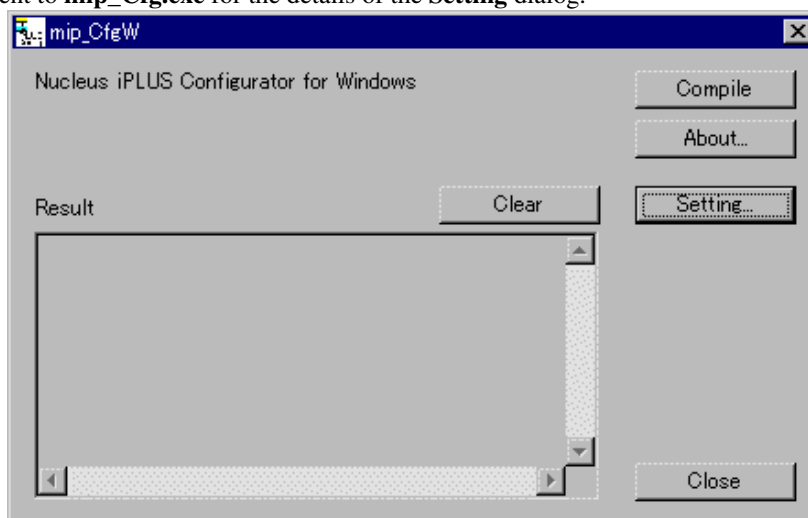


Figure 6.1 mip_CFGW.exe Start-up Window

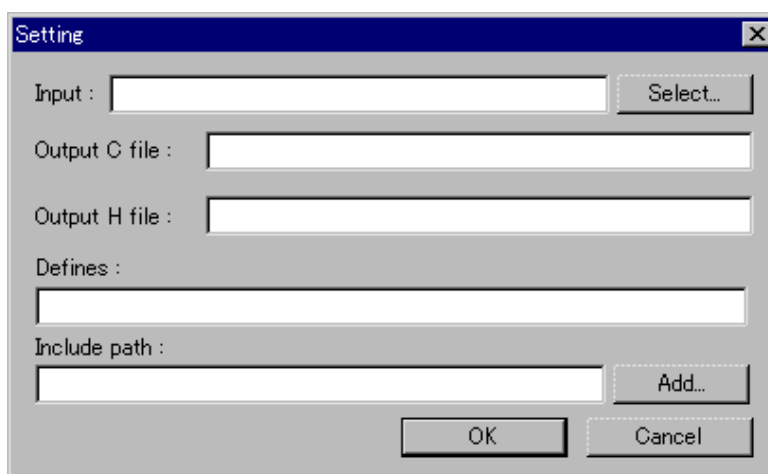


Figure 6.2 Setting dialog of mip_CFGW.exe

6.1 Environmental Variable

The file specified by `#include` is searched by the folder that is defined by `MIPLUS_INC` environmental variables. Furthermore, specifying a number of folders is possible by separating ';' or ','. Priority is higher for the folder that is specified by the command line argument `/I`.

6.2 Pre-processor Directive

_The Pre-processor Directive below could be used:

`#define/#error/#undef/#if/#include/#else/#ifdef/#endif/#ifndef`

_`#elif/#line/#pragma` that is defined in ANSI C is ignored and will not be processed. Furthermore, `#define` that carries an argument will be treated as an error.

_`##` Directive can no be used.

_First taken by `#include` process such as `#define/#if`, but it does not analyze the static API.

_ As an embedded macro, `MIPLUS_CONFIG` and `CONFIGURATOR` are defined. If you want to take the file by `#include`, and when the character that the configurator does not understand is included, you can avoid this by surrounding it with `#ifndef CONFIGURATOR/#endif`.

6.3 Static API

ATT_INI	Attach Initialize Routine
CRE_CYC	Create Cyclic Handler
CRE_DTQ	Create Data Queue
CRE_FLG	Create Event Flag
CRE_MBX	Create Mailbox
CRE_MPF	Create Partition Memory Pool
CRE_MPL	Create Variable-sized Memory Pool
CRE_SEM	Create Semaphore
CRE_TSK	Create Task
DEF_EXC	Define CPU Exception Handler
DEF_INH	Define Interrupt Handler
DEF_TEX	Define Task Exception Process Routine

Table 6.3-1 A list of Static API that could be used by the Configurator (Alphabetical Order)

For the specific of the Static **API** parameter, please refer to the Service Call Reference section of this manual.

6.4 INCLUDE

_The configurator can not interpret the file specified by `INCLUDE()`. It is been developed as `#include` directive. The data that is defined by the static API that are not yet solved, is considered as it is defined in the file that has been taken from `INCLUDE ()`. How to describe the file name is developed below.

```
INCLUDE("\sample.h");
INCLUDE("\sample.h\");
INCLUDE("sample.h");
It will be #include "sample.h"
```

```
INCLUDE(\ "<sample.h>\");
INCLUDE("<sample.h>");
INCLUDE(<sample.h>);
It will be #include <sample.h>
```

6.5 Command Line Argument

_The following command line argument can be used.

The command line argument that starts with ' - ' or ' / ' is an option, and the following could be used:

`/Fcfilename`

Specify the output C Language file. Do not put a space between `/Fc` and filename.

When there is no specification, use kernel_cfg.c as the file name.

/Ffilename

Specify the output header file. Do not put a space between /Fh and filename. When there is no specification, use kernel_id.h as the file name.

/C

Analyze under the condition where extensional comment (‘//’) is not used.

/Ddefine

Specify the definition. More than one definition is possible by separating with ‘, ‘.

/v

Indicate the version information.

/s

Indicate the start message.

/Ifolder

Specify the include folder. Do not put a space between /I and folder. Specifying more than one folder is possible by separating ‘; ‘ or ‘, ‘.

The command line argument that does not start with ‘ – ‘ or ‘ / ‘ is considered as configuration file name. When the configuration file name is not specified, system.cfg file name is loaded.

6.6 Disable the Unnecessary Data Area for Management

Nucleus μ PLUS requires data area for management, and the configurator outputs this. The configurator outputs this even if the area is not used, because it is necessary when you use it by linking the object, even when you don’t use it as a library. When this output is not necessary, defining can disable the output of the unnecessary data area #define MIPLUS_NOLINK.

However in this case, rebuilding of the kernel is necessary by changing the kernel setting that is defined in kernel.h. Please refer to 7.7 **Determining Unnecessary Process**, for determining unnecessary process and management data area.

6.7 User Setting of the Data Area for Management

_Configurator must create a file that could secure at least a minimum area, which is needed by the object that is specified by the configuration file.

_When the data area size for management is secured in the file that has been developed either by the configuration file or by #include directive, described in the configuration file, configurator does not output the definition of the secured data area size for management. The following are such definitions.

TMAX_TSK_OBJ, TMAX_MBX_OBJ, TMAX_DTQ_OBJ, TMAX_SEM_OBJ,
TMAX_FLG_OBJ, TMAX_MPL_OBJ, TMAX_CYC_OBJ

When the user tries to define these, it must be described either in the configuration file or in the file that is developed by #include directive, because the area needs to be secured in the created C file (kernel_ofg.c is the standard).

_When TSZ_KERNEL_POOL_SIZE is defined, its size is used as the operational area of the kernel. When it is not defined, it will be the value (102400 Byte) that is defined in TSZ_KERNEL_POOL_SIZE_DEFAULT. Furthermore, if TSZ_KERNEL_POOL_SIZE is smaller than 50, it operates with no operational area of the kernel, since the system cannot even secure the portion necessary for managing the operational area. Moreover, the operational area of the kernel is not used when the configurator automatically creates it.

_When TSZ_KERNEL_POOL_LOCAL is defined, the operational area of the kernel is created in the local. When it is not defined, the operational area of the kernel is secured by using first_available_memory that is hand over by Application_Initialize ().

_When MIPLUS_NOERROR_CHECKING is defined, the error checking of the kernel initialization process may be disabled.

6.8 Others

_Configurator and Configurator for Windows might be modified without notice.

6.9 Configuration File Sample

```

/*****
/* System Configuration File          */
/* Sample                            */
/*                                  */
*****/
INCLUDE("demo_m.h");

#define STACK_SIZE 2048
#define PRIORITY_DEF 1

CRE_TSK(TASK_MAIN, { TA_HLNG|TA_ACT, 1, task_main, PRIORITY_DEF, STACK_SIZE, NULL });

CRE_TSK(FLG_TASK1, { TA_HLNG|TA_ACT, 0, task_flg1, PRIORITY_DEF, STACK_SIZE, NULL });
CRE_TSK(FLG_TASK2, { TA_HLNG|TA_ACT, 0, task_flg2, PRIORITY_DEF, STACK_SIZE, NULL });
CRE_TSK(FLG_TASK3, { TA_HLNG|TA_ACT, 0, task_flg3, PRIORITY_DEF, STACK_SIZE, NULL });

CRE_TSK(MBX_TASK1, { TA_HLNG|TA_ACT, 0, task_mbx1, PRIORITY_DEF, STACK_SIZE, NULL });
CRE_TSK(MBX_TASK2, { TA_HLNG|TA_ACT, 0, task_mbx2, PRIORITY_DEF, STACK_SIZE, NULL });
CRE_TSK(MBX_TASK3, { TA_HLNG|TA_ACT, 0, task_mbx3, PRIORITY_DEF, STACK_SIZE, NULL });

CRE_FLG(ID_FLG1, { TA_TFIFO|TA_CLR|TA_WMUL, 0 });
CRE_FLG(ID_FLG2, { TA_TFIFO|TA_WMUL, 0 });

CRE_MBX(ID_MBX1, { TA_TFIFO, 3, NULL });
CRE_MBX(ID_MBX2, { TA_MPRI, 3, NULL });

CRE_SEM(ID_SEM1, { TA_TFIFO, 0, 5 });
CRE_SEM(ID_SEM2, { TA_TPRI, 0, 5 });

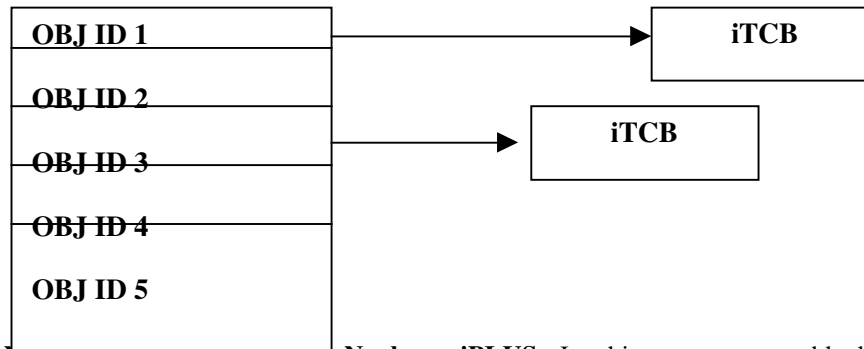
```


7 Internal Structure

7.1 Object Management

The application side prepares the object management area of the kernel. By using the configurator, it outputs the code necessary for the area to be secured. This area is registered in the object list (the pointer arrangement to the object management), when you register the object to the kernel.

`_KERNEL_tsk_list[TMAX_TSK_OBJ]`



Nucleus PLUS is the basis for **Nucleus μ PLUS**. Its object management block must be related, for the functions that are same in **PLUS** and **μ PLUS**.

The table below briefly shows how **μ PLUS** calls the **PLUS** function.

The function of Nucleus μ PLUS	
Task	Use PLUS Task
Semaphore	Use PLUS Semaphore
Event	NEW (different from PLUS 's event)
Data Queue	Use PLUS Queue
Mailbox	NEW
Partition Memory	Partition Memory of PLUS
Variable-sized Portion Memory	Variable-sized Portion Memory of PLUS
Time Management	Time Management of PLUS
Task Exceptional Handler	Use PLUS Signal Process with Modification

Figure 7.1-1 How Nucleus μ PLUS functions have been implemented

Also, the creating information of each object including the task, the kernel use it as working area, so during when the object is effective, it has to be kept. In other words, creating information used by `cre_xxx()` cannot be omitted. Also, these areas must be placed in the **RAM** area.

7.2 Task Management

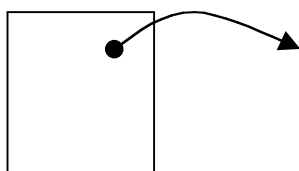
Since **Nucleus μ PLUS** is created by using **Nucleus PLUS** as the basis, **Task Control Block (TCB)** is basically been used; however because of the necessity of having other information, **iPLUS TCB (iTCB)** was created. In order to relate this to **TCB** for **Nucleus PLUS**, the pointer for **iTCB** has been set to the member called `tc_system_reserved_3` in **Nucleus PLUS TCB**.

Furthermore, **TCB** area is included in **iTCB** area.

iPLUS Task Management (Pointer Arrangement)

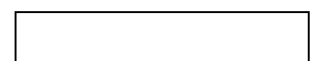
(Pointer)

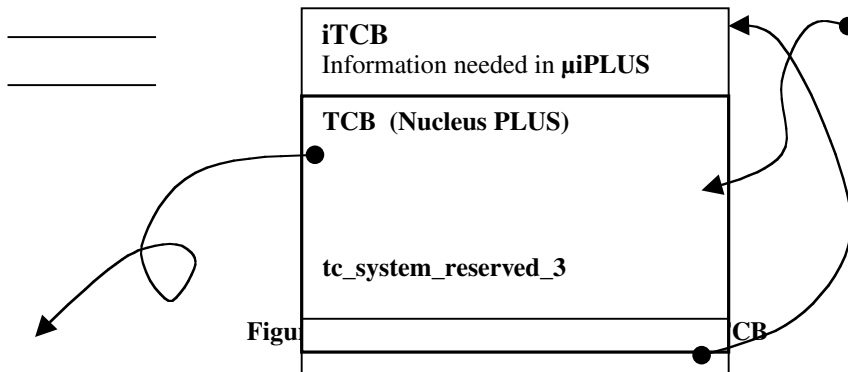
`_KERNEL_tsk_list[TMAX_TSK_OBJ]`



PLUS Task Management

`TCD_Created_Tasks_List`





iPLUS Task Control Block (iTCB)

```

ATR    tskatr;          /* Attribute Task */
VP_INT exinf;           /* Task Extension information */
FP     task;            /* Activate task address */
PRI    itskpri;         /* Priority of the task while activating */
SIZE   stksz;           /* Stack size of the task (# of bytes) */
VP     stk;             /* Starting address of the stack are of the task */
/* ---- Below are the criteria originally added for Nucleus µiPLUS */
VP_INT stacd;           /* Nucleus µiPLUS Task Start Code */
UINT   staflag;         /* Nucleus µiPLUS Task Activate Flag */
T_RTsk rtsk;           /* Nucleus µiPLUS Task information */
NU_TASK ntcb;          /* Nucleus PLUS TCB */
STAT   ret_stat;        /* Status on can_wait */
NU_SEMAPHORE slpsem; /* Semaphore for internal usage by tslp_tsk */
NU_SEMAPHORE dlysem; /* Semaphore for internal usage by dly_tsk */
T_DTEX dtex;           /* Task Exceptional Handler */

```

The field in the shaded area above is the field set by the application when the task is created. It has a semaphore inside, because of **tslp_tsk** and **dly_tsk**. These are the services that disable the current task by having a time out, and also a wait function. **Nucleus PLUS** has the similar function called **NU_Sleep()**, but it does not have a function to cancel the task from waiting. This is why the implementation is in such a way that these services wait until they obtain the semaphore.

7.3 Delayed Execution

In order to protect the internal of the **Operating System**, **Nucleus PLUS** uses a function called Protect that is similar to the semaphore, and does the mutual exclusion by having a thread against the resource that is protected. Because of this, the interrupt latency within the kernel is short, but the service call cannot be used during the interrupt process. In **Nucleus PLUS**, if you want to issue a service call from the interrupt process, you have to activate the **HISR (High Level Interrupt Service)** that has a higher priority than the task thread, and the service call must be issued from this **HISR**. In **Nucleus µiPLUS** service call, **ixxx_yyy** service call that is been called from the interrupt handler will activate the **HISR** from **Nucleus PLUS**. To specify this process:

1. **ixxx_yyy** records the parameter when the service call is issued to the buffer of its corresponding service.
 2. Activate the **HISR**
 3. After the interrupt, **HISR** executes, and the parameter is obtained from the buffer.
 4. Issue the Service Call
- These are the steps in processing.

iact_tsk	Activate Task
----------	---------------

iwup_tsk	Wakeup Task
irel_wai	Forcefully release Wait status
irotd_rdz	Rotate task priority
iras_tex	Request task exception handling
ipsnd_dtq	Send to data queue (polling)
ifsnd_dtq	Forcefully send to data queue
isnd_mbx	Send to mailbox
iset_flg	Set event flag
isig_sem	Return semaphore resource

Figure 7.3-1 Delayed execution service call

7.4 Ready Queue

All of the tasks that are in Ready status are been connected to the TCB pointer array called **TCD_Priority_List**, by the order of its priority.

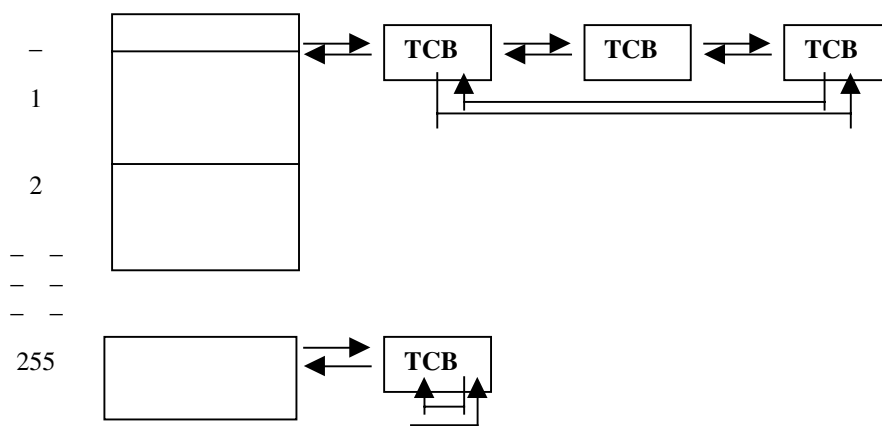


Figure 7.4-1 Ready Queue

In the above figure, the task that corresponds to the top **TCB** of the Priority 1 (array 0) is been activated. Since the activating **TCB** is been pointed from the system variable called **TCD_Current_Thread**, you may know the **TCB** of the activating task by looking at this variable.

7.5 Scheduling Function

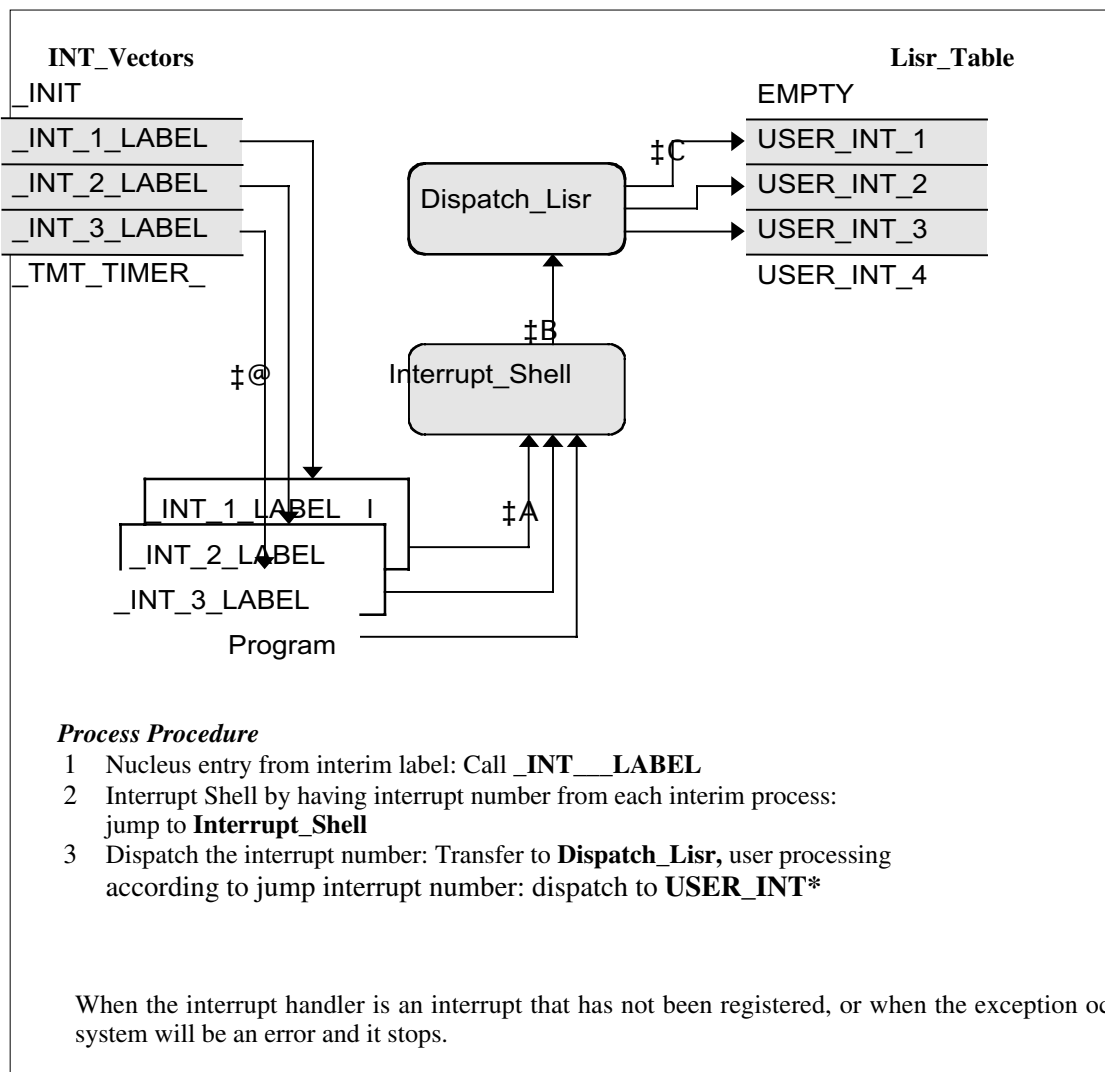
Scheduler Module is a **Nucleus PLUS** file, described in assembler. The task that is to be activated is dispatched by the routine called **TCT_Schedule**. **TCT_Control_To_System** is a function that saves the context of the task to the stack and transfers the process to the scheduler during the task been activated. On the other hand, **TCT_Control_To_Thread** transfers the control to the dispatched task from the

activated scheduler.

7.6 Interrupt Behavior

In external and internal interrupt process, the **OS** once accepts it and calls the interrupt handler, which the user registers, after the saving of the context takes place. Then it calls the scheduler when the interrupt ends, and dispatches if there is a task that is enable to be activated. A portion of interrupt routine is directly described in the interrupt vector table (**INT_Vectors**), which holds the address to the program that bears each interrupt and exception process, as defined in **Nucleus** interrupt process. Examples are the entry during reset, and timer interrupts routine. However, most entries jump to the interim entry routine for processing each interrupts that is already been prepared by **Nucleus**. Therefore, it once goes through the **Nucleus** context saving process, then it calls each interrupt routine that the user registers, from the interrupt dispatcher.

The handler defined by the user **DEF_EXC,DEF_INH** No difference between these two, internally_, is entered in **Lisr_Table** of the diagram below.



7.7 Deterring Unnecessary Process

When there are sets of service calls that are unnecessary, the kernel size can be made smaller by deterring those process and management data. The following definitions are included in **kernel.h**. When the definition is valid, the appropriate sets of service calls process and management data will not be included in the objects. When modification is made, the kernel has to be re-built.

MIPLUS_NOLINK_DTQ	Data queue
MIPLUS_NOLINK_MPF	Fixed-sized memory pool
MIPLUS_NOLINK_MPL	Variable-sized memory pool
MIPLUS_NOLINK_CYC	Cyclic handler
MIPLUS_NOLINK_MBX	Mailbox
MIPLUS_NOLINK_FLG	Event flag
MIPLUS_NOLINK_SEM	Semaphore

7.8 Internal Data Size

The following definition is included in kernel.h, as the internal data size that the kernel uses.

TSZ_IDTQ	The number of data areas, which ipsnd_dtq() and ifsnd_dtq() use for delayed execution.
TSZ_ISEM	The number of data areas, which isig_sem() uses for delayed execution.
TSZ_IFLG	The number of data areas, which iset_flg() uses for delayed execution.
TSZ_ITSK	The number of data areas, which iact_tsk() uses for delayed execution.
TSZ_IATSK	The number of data areas, which iwup_tsk() and irel_wai() use for delayed execution.
TSZ_IROT	The number of data areas, which irot_rdq() uses for delayed execution.
TSZ_ITEK	The number of data areas, which iras_tex() uses for delayed execution.
TSZ_IMBX	The number of data areas, which isnd_mbx() uses for delayed execution.
MIP_SYSH_STACK_SIZE	The stack area size used by ext_tsk().
TSZ_ISR_STACK_SIZE	The stack area size used by ixxy_yyy(), which is other than ext_tsk().
H_EXT_TSK_PRMS	The number of data areas, which ext_tsk() uses as delayed execution.
TSZ_MIP_SYSTEM_MEMORY	System_Memory is the size of the global variable. When this definition is invalid, System_Memory is not created.

7.9 C++

When C++ language is used in the development, there are couple things you have to watch. Please refer to the following warnings.

7.9.1 Constructor of the Global Object

Nucleus PLUS and Nucleus μ PLUS uses their own initializing process of the application. These are one of the porting criteria.

For C++, the compiler standard process is normally used in the constructor process of the global object; however, in Nucleus PLUS and μ PLUS porting, some do not use the compiler standard start-up routine. In this case, calling of the constructor process of the global object must be added. Also, the initialization of the OS is not done at the timing where the constructor of the global object is activated, so the constructor cannot do the process related to OS internally. This problem will not occur when the constructor of the object is created statically. Please refer to the compiler manual that you are using for the constructor process of the global object.

7.9.2 new/delete operator

new/delete operator will be dependant to compiler functionality. When the memory secure/release function of **new/delete** is not reentry, **Nucleus PLUS/ μ PLUS** cannot use the normal **new/delete** operator. Also, when **new** is used, initializing process may have to be added.

However, you can overwrite your own **new** operator and use **get_mpl()/NU_Allocate_Memory()**, so modification according to the application is possible. Please refer to the compiler manual that you are using for **new/delete** operator specification.

Appendix

Maximum data value of the resources

Maximum value of each object	Infinite
Priority of the task	1_255
Message Priority of the mailbox	1_255
Maximum activating request (act_cnt) queing number	255
Maximum activating request (wup_cnt) queing number	255
Maximum forceful wait request (sus_cnt) nest number	255
Maximum resource number of the semaphore	65535



Nucleus μ iPLUS

μ ITRON4.0 Specification
User's Manual
English Version 1.04

April 26, 2001

Grape Systems Inc.
Queen's TowerB 8F, 2-3-3, Minatomirai Nishi-ku, Yokohama, Japan 220-6108
Basic Software Division (Support Group) : +81-45-222-3762
support@nucleus.grape.co.jp
<http://www.grape.co.jp/>

Accelerated Technology
720 Oak Circle Drive, East
Mobile, AL 36609
800-468-6853
334-661-5770
334-661-5788 (fax)
support@atinucleus.com
<http://www.atinucleus.com/>